arXiv:2603.28118v1 [cs.DS] 30 Mar 2026

偏序集中理想与反链的常数延迟格雷码枚举

Sofia Brenner*和 Ji˘r´ı Fink†

摘要

我们提出一种算法，能够以格雷码顺序、常数延迟地枚举输入偏序集的所有理想；也就是说，连续访问的理想之间至多相差三个元素。这解决了 Pruesse and Ruskey 提出的一个长期未解问题，并改进了 Pruesse and Ruskey、Squire、Habib、Medina、Nourine 和 Steiner 以及 Abdo 先前提出的算法。运用相同的技术，我们还得到一种算法，能够以常数延迟枚举输入偏序集的所有反链，并且连续访问的反链之间至多相差三个元素。

作为一项关键的技术要素，我们为递归算法引入了一种新的基于势能的分析框架，称为金字塔方法。我们证明，该方法涵盖了 Uno 的推出方法。除本文的应用之外，金字塔方法还是一种用于分析递归算法的通用框架，因此可能具有独立的研究价值。

1 引言

1.1 组合枚举

“枚举算法 ”的目标是将有限集合 X 中的所有对象各枚举一次。集合 X 通常以隐式描述给出，算法的任务是计算出其中的元素。显然，我们追求的是能够高效枚举 X 且只需少量内存的算法。我们关注组合对象的枚举，例如图、偏序集的子结构，或排列的子集。

组合结构的枚举历史悠久。枚举一个集合 X 的最基本算法是将其 X 视为所有长度为 n 、取自字母表 Σ的字符串组成的集合。该算法以非确定性方式为 Σ 所有 n 位置选择一个字母，并在生成的字符串属于 X时接受该字符串。Floyd [和 Flo67]的一篇早期论文将这一思路转化为确定性回溯

*卡塞尔大学自然科学与数学系，德国。现地址：查理大学应用数学系，捷克共和国布拉格。电子邮箱：sbrenner@mathematik.uni-kassel.de

†查理大学理论计算机科学与数理逻辑系，捷克共和国布拉格。电子邮箱：fink@ktiml.mff.cuni.cz

算法，并采用剪枝技术和选择合适字符串的策略来提升性能。Read 和 Tarjan [RT75] 枚举了图子结构，例如生成树、环和各种路径。后来，这种方法被称为二分划分法或手电筒搜索（例如，参见 [MM24]）。Fukuda 和 Matsui [FM94]在时间 $\mathcal{O}(m|X|)$ 和空间 $\mathcal{O}(nm)$ 内枚举了二分图中的所有完美匹配。Avis 和 Fukuda [AF96]提出了反向搜索的概念。他们将其用于枚举平面上由 n 个点构成的所有三角剖分、图的生成树以及有向无环图的拓扑排序。近期的研究包括 Conte、Grossi、Marino、Uno 和 Versari [CU19, CGM+22]提出的邻近搜索框架，他们利用该框架枚举给定图中的极大子图或极大树等对象。Merino 和 M¨utze [MM24] 提出了一种通过组合优化列出 $0/1$-多面体所有顶点的框架。由此产生了多种组合对象的枚举算法，包括拟阵的基和独立集、生成树、森林、匹配或顶点覆盖。Fink [Fin25] 推导出一种用于列出完美匹配的常数延迟算法。

对于许多枚举算法，尤其是组合枚举算法，所枚举集合 X 的大小相对于输入规模呈指数级增长。因此，人们更倾向于衡量连续访问的两个元素之间所花费的计算时间。在计算模型中，指令是指在常数时间内执行的一项基本操作，例如确定性随机存取机或指针机。令 $t(i)$ 表示访问第 i 个元素之前执行的指令数。枚举算法的（最坏情况）延迟 是连续访问的两个元素之间所花费时间的最大值，即 $\max_{i\in\{1,\dots,|X|-1\}}\{t(i+1)-t(i)\}$。显然，最理想的结果是获得常数延迟算法。这类算法通常称为无循环算法。在许多应用中，两个输出之间的时间差异很大，因此人们常考察平均值。枚举算法的摊还延迟 是连续两个输出之间所花费的平均计算时间，即算法的总计算时间除以其输出数量。请注意，有些作者将其称为平均延迟，并将算法的摊还延迟定义为 $\max_{i\in\{1,\dots,|X|\}} t(i)/i$。本文也将给出该基数的界。

在许多情况下，我们要求枚举算法的连续输出之间只相差一个小变化，这种变化称为翻转。翻转的定义取决于具体问题。符合这一条件的枚举元素列表称为组合格雷码。集合 X 上的组合生成问题可以通过其翻转图来描述。翻转图的顶点是 X 中的元素，两个元素只要相差一次翻转，就由一条边相连。因此，寻找组合格雷码就相当于在翻转图中寻找一条哈密顿路径。经典的二进制反射格雷码 [Gra53] 列出了所有固定长度的二进制串 n，且任意两个相邻字符串仅相差一个比特。Steinhaus-Johnson-Trotter 算法 [Tro62, Joh63, Ste64] 列出了长度为 n 的排列，使得相邻列出的两个排列仅相差一次相邻值的换位。近年来，组合格雷码领域涌现了大量研究成果。其中一条主要研究脉络是 M¨utze 及其合作者提出的排列语言框架（参见 [HHMW20]）。他们证明了许多类组合对象都存在组合格雷码，例如某些类的模式避免排列 [HHMW20], 、弱序的商 [HM21], 、矩形剖分 [MM23a],、消去

树 [CMM24], 无环定向 [CHMM23], 二叉树 [GMN24], 或超平面排列的区域图 [BCM+26]。在许多情况下，格雷码的构造是明确的，可以转化为算法，但并非所有情况下都已知无循环算法。关于组合格雷码的全面综述，请参阅 M¨utze 的综述 [M¨ut23]。

1.2 偏序集的枚举算法

偏序集是一个有序对 $(P, \preceq)$，由集合 P 和定义在 P 上的偏序 $\preceq$ 组成；偏序是一种自反、反对称且传递的关系。本文只考虑有限偏序集。如果关系 $\preceq$ 可由上下文确定，我们就将偏序集 $(P, \preceq)$ 简记为 P。对于 $u, v \in P$，若 $u \preceq v$ 且 $u \neq v$，则记作 $u \prec v$，并称 u 小于 v，或称 v 大于 u。如果 $u \preceq v$ 和 $v \preceq u$ 均不成立，则称元素 u 和 v 不可比。此时，我们称集合 $\{u, v\}$ 为一个不可比对。如果 $u \prec v$ 且不存在满足 $u \prec z \prec v$ 的 $z \in P$，则称 v 覆盖 u，或称 u 和 v 之间存在覆盖关系。表示偏序集的一种实用工具是其哈斯图。哈斯图是在平面上绘制的图，其顶点是 P 的元素。如果 v 覆盖 u，我们就在 u 和 v 之间连一条边，并将 v 画得比 u 高。示意见图 1(a)。$x \in P$ 的上集和下集分别由所有满足 $x \preceq u$ 或 $u \preceq x$ 的元素 $u \in P$ 组成。若 I 是 P 的一个子集，且它是向下封闭的，即对于所有 $u \in I$，u 的下集都包含于 I，则称 I 为一个理想。P 的理想集合记为 $\mathcal{I}(P)$。P 中的链是由 P 中两两可比的元素组成的子集；反链是由两两不可比的元素组成的子集。恰含 k 个元素的反链称为 k-反链。

偏序集在组合学和算法领域都占有重要地位，因此其算法方面已得到广泛研究。例如，研究表明，偏序集中的各种计数问题都是 #P 完全的，例如计算反链的数量 [PB83]（因此也就是理想的数量，因为每个偏序集中的理想和反链数量相同），或给定偏序集的线性扩张数量 [BW91]。Pruesse and Ruskey [PR94] 提出了一种枚举输入偏序集线性扩张的算法，使得连续枚举出的扩张之间相差一次或两次相邻换位。Canfield 和 Williamson [CW95]将其转化为无循环算法。

相比之下，理想的无循环枚举长期以来一直是一个悬而未决的问题。关于这一问题的讨论可参见 [PR93、PR94、HMNS01、M¨ut23、Pow25] ，其中提供了综述。对于偏序集中的理想，目前最快的枚举算法无论是按格雷码顺序列出，还是按任意顺序列出，延迟均为对数级。对于 $k \in \mathbb{N}$，若连续访问的两个理想之间相差至多 k个元素，我们就称这种列举为-格雷码（即它们的对称差大小至多为 k ）。在文献中，格雷码顺序通常定义为 2-格雷码，即连续列出的理想相差至多两个元素。换言之，一个理想可通过移除、添加或交换一个元素，从另一个理想得到。在我们的算法中，我们构造了一个 3-格雷码，即连续访问的理想之间最多相差三个元素。请注意，这不会改变算法的渐近时间复杂度。

早期算法 [Law79、SB78、PB83]列出大小为 n 的偏序集的理想，延迟为 $\mathcal{O}(n^2)$。Steiner $\mathcal{O}(n$给出了可能是首个平均延迟为 [Ste86]的算法。Pruesse and Ruskey [PR93] 为一般偏序集提出了一种（2-）格雷码枚举算法，运行时间为 $\mathcal{O}(|\mathcal{I}(P)|n)$

面向一般偏序集，采用针对反拟阵的更一般框架。然而，他们的算法会将每个元素列出两次。Medina 和 Nourine [MN94]提出了一种摊还延迟为$\mathcal{O}(d^+(P))$的算法，其中 $d^+(P)$ 表示 P的哈斯图中的最大出度。Squire [Squ95]（另见 [Pow25]）提出了一种平均延迟为 $\mathcal{O}(\log n)$ 的算法，但列出的元素并非按格雷码顺序排列。Habib、Medina、Nourine 和 Steiner [HMNS01] 在其枚举分配格的框架中提出了另一种算法，该算法按格雷码顺序列出理想，摊还延迟为 $\mathcal{O}(n)$。Abdo [Abd09, Abd10, Abd13]又提出了其他按格雷码顺序枚举理想且摊还延迟为 $\mathcal{O}(n)$ 的算法。Merino 和 M¨utze [MM23b, MM24] 指出，他们用于枚举 $0/1$-多胞体顶点的更一般框架，可将偏序集理想的枚举算法作为特例导出。然而，该算法的延迟为 $\mathcal{O}\big(n^4 \log n\big)$。对于反链，他们的框架也能导出一种算法，不过其延迟为 $\mathcal{O}\big(n^{2.5} \log n\big)$。对于某些特殊的偏序集类，例如森林偏序集 [KR93] 或区间偏序集 [HNS97]，已知存在平均延迟为常数的算法。

上述许多算法，例如 Squire 和 Abdo 提出的算法，都依赖一种递归策略，将输入偏序集 P 的理想分成两个子集。一类是包含固定元素 $x \in P$（因而也包含其下集）的理想，另一类是不包含 x（因而也不包含其上集中的任何元素）的理想。主要区别在于固定元素 x的选择。如果输入偏序集是一条链，很容易看出，这种策略至少会产生对数延迟。因此，要实现常数延迟算法，就需要采用不同的递归策略。

最后，我们提及上述问题的一个变体：生成偏序集中固定大小的理想。在这种情况下，依次访问的理想之间仅相差一个元素的替换。多年来，这一变体受到了广泛关注。在 [KR93], 中，作者未给出证明便指出，他们用于生成森林偏序集中所有理想的算法，也可用于生成固定大小的理想。在 [PR93], 中，这一问题随后得到了专门研究。Wild [Wil14] 提出了一种具有 $\mathcal{O}\big(n^3\big)$ 延迟的算法。最近，Coumes、Bouadi、Nourine 和 Termier [CBNT21], 利用这一方法枚举了所谓的天际线群，它们用于决策。他们证明，这些群对应于某些偏序集中固定大小的理想。在该论文中，他们给出了一种算法，能够以 k 延迟访问偏序集中大小为 $\mathcal{O}\big(\omega^2\big)$的所有理想，其中 ω 表示输入偏序集的宽度。关于固定大小理想枚举的更多应用，例如码和覆盖阵列，可参阅 Powers 的博士论文 [Pow25]。

1.3 我们的研究成果

本文旨在提供一种算法，以常数摊还延迟枚举偏序集理想和反链。本文分为两部分：第一部分中，我们提出一种势能分析方法——金字塔条件，用于分析枚举算法的摊还时间复杂度。第二部分中，我们介绍一种按 3-格雷码顺序枚举偏序集中理想的枚举算法。我们利用金字塔条件证明该算法具有常数延迟。通过相同的技术，我们还得到了一种用于枚举反链的常数延迟算法。

1.3.1 复杂度分析和金字塔条件

我们首先提出势能分析的一种新变体，称为金字塔方法。我们主要关注如何分析组合枚举算法。不过，这一概念并非专为这种情形设计。该方法利用了这样一个思想：在枚举算法的递归树中，叶节点对应于连续快速生成输出、且只需较少计算时间的迭代；而内部节点的情况恰好相反。因此，摊还的思路是利用叶节点多出的计算时间，完成树中较高层节点的计算。为此，我们使用一种势能来衡量子节点可以为其父节点节省的计算时间。图2展示了一个简单算法的示例，该算法用于枚举输入图中的完美匹配。

对于一次迭代 X，令 $T(X$表示执行迭代 X所需的指令数， $C(X$表示由 X 递归调用的迭代集合（即子节点）， X而 X' 表示直接访问的元素集合 X。完整定义请参见第2 节。上述思想由以下定义概括：

定义1.1（金字塔条件）。如果存在常数 $\mu > 0$ ，使得对于 $\mathcal{A}$的每次迭代 X ，均有以下结论，则称递归算法 $\mathcal{A}$ 关于势函数 Φ 和时间 $T^\star$ 满足金字塔条件：

$$\sum_{Y \in C(X)} \Phi(Y) + \mu|X'| - \Phi(X) \geq \frac{T(X)}{T^\star}.$$

令 Δ 为递归树中从根节点到叶节点的路径上指令数的上界。本部分的主要结果如下：

定理1.2。令 $\mathcal{A}$ 为一个枚举集合 X 的递归算法，并假设 $\mathcal{A}$关于势函数 Φ 和时间 $T^\star$满足金字塔条件。枚举 X 的总计算时间为 $\mathcal{O}(|X|T^\star - \Phi(X))$，因此 $\mathcal{A}$ 的摊还延迟为 $\mathcal{O}(T^\star)$。 X 的前 k 个元素可在 $\mathcal{O}(kT^\star + \Delta)$时间内访问。

枚举前 k 个元素的时间复杂度（见定理 1.2）中，加法常数 Δ可视为开始时所需的预处理时间，而每个被访问的元素需要 $T^\star$ 额外时间。

现在我们从摊还延迟转向最坏情况延迟分析。众所周知，可以将摊还延迟为 $T^\star$的算法转换为最坏情况延迟为 $T^\star$的算法，但可能需要付出指数级内存消耗的代价。借助 Uno 在一份未发表的技术报告中最初提出的思路 [Uno03], 我们得到以下适用于满足金字塔条件的算法的去摊还结果：

定理 1.3（最坏情况延迟）。设 $\mathcal{A}$ 是一个递归算法，用于枚举集合 X；该集合相对于势函数 Φ 和时间 $T^\star$ 满足金字塔条件。假设 $\mathcal{A}$ 中连续访问的两个元素之间的变化可以用 $\mathcal{O}$（1）内存存储，并且这些变化可以

ˆ在 $\mathcal{O}(T^\star)$ 时间内应用。那么，存在一个算法 $\mathcal{A}$，它按照与 $\mathcal{A}$ 相同的顺序枚举 X，最坏情况延迟为 $\mathcal{O}(T^\star)$，同时需要 $\mathcal{O}(\Delta/T^\star)$额外空间，并需要 $\mathcal{O}(\Delta)$额外时间进行初始化。

虽然不存在 Δ的一般上界，但在许多应用中，递归树的深度以及从根节点到叶节点的最长路径上的指令数，都受输入规模的多项式界限制。在这种情况下，算法 $\mathcal{A}'$（定理 1.3 ）所需的额外空间以及初始化时间都是输入规模的多项式。一个典型的例子是简单的枚举算法，例如生成树枚举算法：每次迭代都从输入图中选取一个对象，例如决定是否纳入一条边。显然，递归树的深度以及从根节点到叶节点的路径上的迭代次数，都至多与输入规模呈线性关系。

与推出方法的比较 Uno [Uno15]提出了一种分析递归算法摊还时间复杂度的方法，即所谓的 推出方法。他利用该方法证明，许多组合枚举算法（例如枚举消去序、匹配森林、连通诱导子图或给定图中的生成树）具有常数摊还延迟。我们证明，金字塔方法涵盖了推出方法：

定理 1.4。 如果递归枚举算法 $\mathcal{A}$ 满足 *Uno* 的推出条件，且时间复杂度为 $T^\star$（即该方法给出摊还时间复杂度 $T^\star$），那么存在一个势函数 Φ，使得 $\mathcal{A}$ 相对于 Φ 和时间 $T^\star$ 满足金字塔条件。

定理 1.4 指出，任何通过推出方法获得摊还延迟 $T^\star$ 的算法，都可以用金字塔方法进行分析，从而得到相同的摊还延迟上界 $T^\star$ 。与此同时，我们用于枚举偏序集中的理想和反链的算法，就是一类无法用推出方法证明其具有常数摊还延迟、但可以用金字塔方法证明的算法。我们在示例 3.13中给出了一个偏序集的例子。这表明金字塔方法严格强于推出方法。

直观地说，在推出方法中，每个子节点 Y 都会向其父节点发送 $\mathcal{O}(T(Y))$。相比之下，金字塔方法是一个适用范围更广的框架，因为它允许在子节点与父迭代之间转移任意数量的计算时间。更准确地说，当且仅当势函数具有 $\Phi(X) = \mathcal{O}(T(X))$ 的形式时，推出条件才成立。在这种情况下，两种方法都可用于证明常数摊还延迟。

1.3.2 偏序集理想的常数延迟枚举

本文的主要结果是针对偏序集中理想的一种递归枚举算法，该算法具有常数延迟，并且所列出的理想中，连续访问的两个理想至多相差三个元素。我们首先介绍一个以任意顺序列出理想、且具有常数摊还延迟的版本。算法1.1中给出了伪代码描述。随后，我们将展示如何

改进此算法，使其按格雷码顺序访问理想，并调用定理 1.3 以实现常数延迟。

算法 1.1： 枚举偏序集中的所有理想（基础版本）。

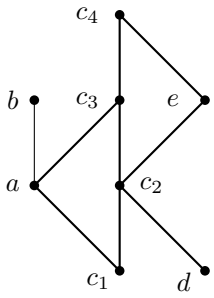
```
过程 枚举(偏序集 P， I数组，其中包含 P之外的元素)
    如果 P = ∅，则
        访问(I)
    else
        在 P中找到一条最长链 C
        构造辅助集 L_2, ..., L_k 和 S_0, ..., S_{k-1}
        P' ← S_0 // P' = P_0
        枚举(P', I) // 枚举 P_0 中的理想
        对于 i ← 1, ..., k do
            P' ← (P' \ L_i) ∪ S_i // P' = P_i
            I ← I ∪ L_i ∪ {c_i}
            枚举(P', I) // 枚举 P_i中的理想
        I ← I \ P // 移除本次迭代添加到 I 中的所有元素。
过程 主程序(偏序集 P)
    重新标记 P，使得 u ⪯ v 蕴含 u ≤ v
    枚举(P, ∅)
```

递归结构 与现有算法相比，我们的算法的一个主要区别在于其递归结构。不同于以往算法根据理想是否包含某个固定元素 $x \in P$将理想集划分为两部分，我们利用一条最长链来划分理想。这样，当输入偏序集为一条链时，我们便能够获得常数延迟，而采用先前使用的递归技术则无法做到这一点（例如，参见 [Squ95、Abd13]）。

设 C是 P的一条最长链，并将 $C = \{c_1, \ldots, c_k\}$写作 $c_1 \preceq \cdots \preceq c_k$。算法 1.1 的递归将 P的理想划分为 $k+1$个集合 $\mathcal{I}_0(P) \ldots, \mathcal{I}_k(P)$，具体取决于理想中包含的 C 的最大元素（或者不存在这样的元素）。形式上，对于 $i > 0$，集合 $\mathcal{I}_i(P)$由 P的理想 I组成，其中 c_i是 $I \cap C$中的极大元素。集合 $\mathcal{I}_0(P)$包含 P 中不含 C的任何元素的理想。

现在，我们定义偏序集 P 的子偏序集 P_0、……、P_k，并在这些子偏序集上进行递归。对于 $i = 0$、……、k，偏序集 P_i 是偏序集 P 的子偏序集，由所有满足 $u \not\preceq c_i$ 和 $u \not\succeq c_{i+1}$ 的元素 $u \in P$ 构成（约定对于所有 $u \in P$，$c_0 \preceq u \preceq c_{|C|+1}$）。递归所依据的关键洞见是，$\mathcal{I}_i(P)$ 中的理想与 P_i 的理想之间存在双射（见引理 3.3）。该双射将理想 $I \in \mathcal{I}_i(P)$ 映射为 $I \setminus D_i$，其中 D_i 表示 c_i 的下集。因此，要得到 $\mathcal{I}_i(P)$ 中的理想，我们只需递归计算 P_i 中的理想，并将集合 D_i 加入每个理想。图 1 展示了 P_i 的定义，以及将理想划分为集合 $\mathcal{I}_i(P)$ 的过程。

(a)



$P_0 = \{d\}$ $S_0 = \{d\}$

$P_1 = \{a, b, d\}$ $S_1 = \{a, b\}$ $L_1 = \emptyset$

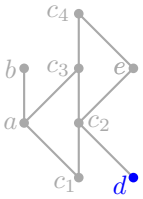
$P_2 = \{a, b, e\}$ $S_2 = \{e\}$ $L_2 = \{d\}$

$P_3 = \{b, e\}$ $S_3 = \emptyset$ $L_3 = \{a\}$

$P_4 = \{b\}$ $S_4 = \emptyset$ $L_4 = \{e\}$

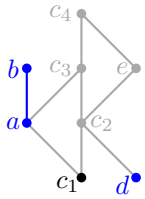
$L_5 = \{b\}$

(b)



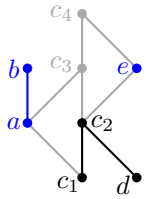
$D_0 = \emptyset$

$D_0 \cup \emptyset = \emptyset$
$D_0 \cup \{d\} = \{d\}$



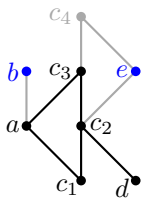
$D_1 = \{c_1\}$

$D_1 \cup \emptyset = \{c_1\}$
$D_1 \cup \{a\} = \{c_1, a\}$
$D_1 \cup \{d\} = \{c_1, d\}$
$D_1 \cup \{a, b\} = \{c_1, a, b\}$
$D_1 \cup \{a, d\} = \{c_1, a, d\}$
$D_1 \cup \{a, b, d\} = \{c_1, a, b, d\}$



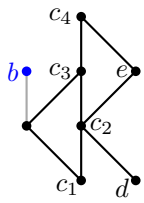
$D_2 = \{c_1, c_2, d\}$

$D_2 \cup \emptyset = \{c_1, c_2, d\}$
$D_2 \cup \{a\} = \{c_1, c_2, d, a\}$
$D_2 \cup \{e\} = \{c_1, c_2, d, e\}$
$D_2 \cup \{a, b\} = \{c_1, c_2, d, a, b\}$
$D_2 \cup \{a, e\} = \{c_1, c_2, d, a, e\}$
$D_2 \cup \{a, b, e\} = \{c_1, c_2, d, a, b, e\}$



$D_3 = \{c_1, c_2, c_3, a, d\}$

$D_3 \cup \emptyset = \{c_1, c_2, c_3, a, d\}$
$D_3 \cup \{b\} = \{c_1, c_2, c_3, a, d, b\}$
$D_3 \cup \{e\} = \{c_1, c_2, c_3, a, d, e\}$
$D_3 \cup \{b, e\} = \{c_1, c_2, c_3, a, d, b, e\}$



$D_4 = \{c_1, c_2, c_3, c_4, a, d, e\}$

$D_4 \cup \emptyset = \{c_1, c_2, c_3, c_4, a, d, e\}$
$D_4 \cup \{b\} = P$

图 1：（a）偏序集 P 的示例，以及子偏序集 P_0、……、P_4，还有集合 S_0、……、S_4 和 L_1、……、L_5，均以其哈斯图表示。（b）集合 D_i 以及 $\mathcal{I}_i(P)$ 中的理想，其中 $i = 0$、……、4。$\mathcal{I}_i(P)$ 中的每个理想都具有 $I' \cup D_i$ 的形式，其中 I' 是 P_i 的理想。P_i 和 D_i 中的元素分别以蓝色和黑色标出。

为了在从 P_i 转换到 P_{i+1} 时高效更新当前处理的偏序集，并计算 c_i 的下集 D_i，我们使用下面定义的辅助子集 S_0、……、S_k 和 L_1、……、L_{k+1}。这些子集与链 C 同时计算。下面将详细介绍这些计算及其时间复杂度。

计算链和更新为了计算最长链，我们利用反链分解推导出一种特殊算法，其时间复杂度为 $\mathcal{O}(n+q)$，其中 q 是不可比对的数量，因为经典拓扑排序算法的时间复杂度 $\mathcal{O}(n^2)$ 不足以完成此任务。然后，我们在时间 $\mathcal{O}(n+q)$ 内计算集合 S_0、……、S_k 和 L_1、……、L_{k+1}。这里，S_i 由所有满足 $u \succ c_i$ 和 $u \not\succ c_{i+1}$ 的元素 $u \in P \setminus C$ 组成（换句话说，c_i 是 C 上小于 u 的最大元素）。对称地，我们将 L_i 定义为满足 $u \prec c_i$ 和 $u \not\prec c_{i-1}$ 的元素 $u \in P \setminus C$ 构成的集合。由于 C 是一条最长链，因此我们有 $S_k = L_1 = \emptyset$。注意，采用此记号，我们有 $D_i = D_{i-1} \cup L_i \cup \{c_i\}$ 和 $P_i = (P_{i-1} \setminus L_i) \cup S_i$。我们使用这些规则更新当前偏序集 P'（在该偏序集上调用递归），以及当前集合 I；集合中包含已添加到正在构造的理想中的元素。

令 $n_i = |P_i|$（$i = 0, \ldots\ldots, k$）。按顺序 $P_0, \ldots\ldots, P_k$处理偏序集时，可以证明，P_i 中的每个元素被添加和删除的次数都是常数，因此，更新所需的变更总数为 $\mathcal{O}(\sum n_i) \leq \mathcal{O}(n+q)$。构造 C 时进行复杂度分析的关键观察是：每个不可比对至多进行一次比较，并且每个元素至多与一个比它小的元素进行比较。计算所有集合 S_i、L_i、D_i 和 P_i时也采用了类似策略。为了分析整个算法的复杂度，我们采用金字塔方法，并使用如下形式的势函数：$\Phi(P) = \alpha + \beta n + \gamma q + \delta t$，其中 $\alpha, \beta, \gamma, \delta \in \mathbb{N}$。这里，$t$ 表示 P中的不可比三元组数量（即 3-反链数量）。

格雷码枚举 算法的基本版本 1.1 以任意顺序枚举理想。我们提出一个改进版本，枚举输入偏序集 P 的理想，使得每一对连续访问的理想至多相差三个元素。这是通过为每次对 P_0、……、P_k的递归调用选择特殊的排序方式以及预先指定的起始理想和终止理想来实现的。利用定理 1.3中的技术，我们可以将其转换为无循环算法。

根据 Pruesse and Ruskey 的一个结果 [PR93]，，偏序集 P 的理想所构成的翻转图中，如果两个理想至多相差两个元素便连接一条边，则该图具有一条哈密顿路径。然而，目前尚不清楚是否能够以常数摊还延迟计算出这样一条路径。我们认为，或许可以改进我们的算法，使连续访问的理想至多相差两个元素，同时保持常数延迟。不过需要注意的是，这不会影响算法的渐近运行时间。

我们总结了偏序集理想枚举方面的结果：

定理 1.5。 算法 *1.1* 枚举偏序集的所有理想 P 在 n个元素上具有常数摊还延迟，同时需要 $\mathcal{O}(n^2)$ 空间和 $\mathcal{O}(n^2)$ 时间进行初始化。对于 $k \in \mathbb{N}$，

前 k 个理想的访问时间为 $\mathcal{O}(k+n(n+q))$，其中 $q \in \mathcal{O}(n^2)$ 表示2-反链在 P 中的数量。

我们可以将算法 *1.1* 改进为一个无循环算法，按格雷码顺序访问 P 的理想；也就是说，任意两个连续访问的理想之间最多相差三个元素。

1.3.3 用于枚举反链的常数延迟算法

采用与处理理想时相同的技巧，我们可以得到一个用于枚举反链的常数延迟算法（算法 1.2）。如上所述，我们可以改进该算法，使其按格雷码顺序列出反链；也就是说，依次访问的反链之间最多相差三个元素。

算法 1.2： 枚举偏序集中的所有反链（基本版本）。

```
过程 枚举(偏序集 P，元素数组 A （在 P之外））
    如果 P 为空 则
        访问(A)
    else
        在 P 中查找最长链 C
        构造 L_2, ..., L_k 和 S_0, ..., S_{k-1}
        P' ← S_0 //  P' = P_1
        枚举(P', A ∪ {c_1})
        对于 i ← 2, ..., k do
            P' ← (P' \ L_i) ∪ S_{i-1} //  P' = P_i
            枚举(P',  A ∪ {c_i})
        枚举(P \ C,  A)
过程 主程序(偏序集 P)
    对元素进行排序，使得 u ⪯ v 蕴含 u ≤ v
    枚举（P,  ∅）
```

定理 1.6. 算法*1.2*枚举偏序集的所有反链 P 在 n个元素上，具有常数摊还延迟，且初始化需要 $\mathcal{O}(n^2)$ 空间和 $\mathcal{O}(n^2)$ 时间，其中 $q \in \mathcal{O}(n^2)$表示2-反链在 P中的数量。前 k 个反链的访问时间为 $\mathcal{O}(k+n(n+q))$。我们可以将算法*1.2*改进为无循环算法，按格雷码顺序访问 P 的反链，即任意两个连续访问的反链至多相差三个元素。

本文结构安排如下：第2节介绍金字塔条件，这是一种用于递归算法势能分析的新方法。第3节介绍一种以常数延迟枚举偏序集中理想的算法。第4节给出一种用于枚举反链的常数延迟算法。

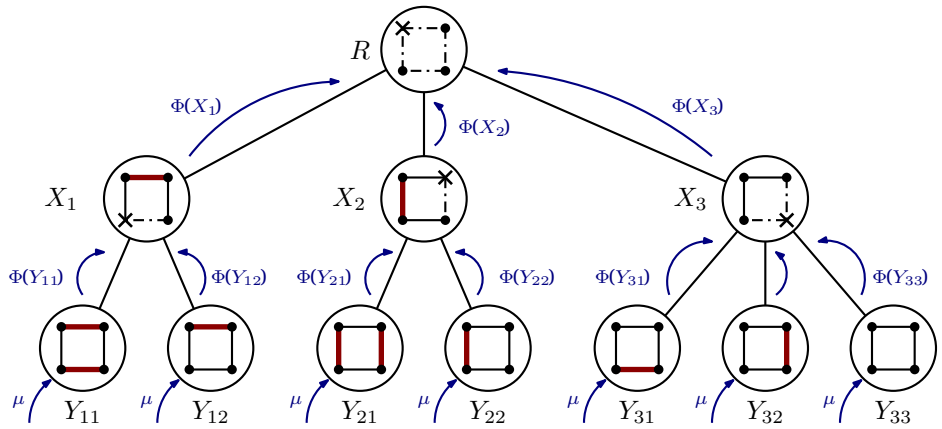


图 2：枚举图中所有匹配的递归树 G （参见示例 2.1）。在每次迭代（树的一个节点）中，选取偏序集 v 的 G 一个顶点（标记为 $\times$）。递归调用对应于选择一条与 v 关联的边，将其加入匹配（以红色标记），或决定不加入与 v 关联的任何边（被排除的边以实心黑色标记）。开始时，G 中的所有边状态均未确定（以虚线表示）。递归树的每个叶节点都对应于 G的一个匹配。

2 组合枚举的势分析

本节介绍一种新的势能分析方法，称为金字塔方法。该方法将用于证明，用于枚举偏序集中理想和反链的算法（算法1.1和1.2）具有常数摊还延迟。不过，我们强调，所介绍的方法可应用于任何枚举算法，因此也可能具有独立的研究价值。

在第 2.1 节中，我们介绍所分析算法的设置。在第 2.2 节中，我们描述金字塔方法。在第2.3节中，我们讨论如何将满足金字塔条件且具有摊还延迟 T^* 的算法转化为最坏情况延迟为 T^* 的算法，前提是元素按格雷码顺序访问。

2.1 递归算法

现在，我们使用与 Uno 类似的术语，来描述将要分析的递归算法的设定。 [Uno15]。设 S 是一个将使用递归算法进行枚举的集合 $\mathcal{A}$。例如，考虑下面这个枚举图中所有匹配的例子。

例 2.1. 我们考虑一种简单的算法，用于枚举输入图中的所有匹配 G（另见 *[Uno15，第 5]*节）。示意见图 *2*。该算法维护一个由匹配中包含的边组成的集合。开始时，该集合为空。在每次迭代中，我们固定偏序集 v 中的一个顶点 G，并令 $e_1, \ldots, e_k$ 为与 v相 *incident* 的边。显然，偏序集 G 的每个匹配至多包含其中一条边。因此，对于每个 $i \in \{1, \ldots, k\}$，函数都会进行一次递归调用

将边 e_i 加入匹配中，另外再进行一次调用，表示不包含边 e_1、..、e_k 中任何一条的情形。

令 $T(\mathcal{A})$ 为表示 $\mathcal{A}$ 中递归调用的递归树。$T(\mathcal{A})$ 中的每个节点都对应 $\mathcal{A}$ 的一次递归函数调用，并可能包含一些额外的非递归计算。递归函数 $\mathcal{A}$ 的一次迭代是指它的一次执行，不包括该迭代递归调用的函数所进行的计算。我们可以将 $\mathcal{A}$ 的各次迭代与其递归树中的节点对应起来，并交替使用这两种表示。每次迭代都关联一个子集 $X \subseteq S$，其中包含在该次迭代或其某次递归调用中访问的元素。我们用集合 X 来标记并识别该迭代。在迭代 X 中，会导出一个划分 $X = X_1 \dot{\cup} \cdots \dot{\cup} X_k \dot{\cup} X'$，其中 X_1、.....、$X_k \neq \emptyset$。这里，X' 表示迭代 X 直接访问的元素集合，该集合可能为空。函数会针对集合 X_1、.....、X_k 递归调用。这些迭代称为 X 的子节点；在递归树中，对应于 X_1、.....、X_k 的节点是对应于 X 的节点的子节点。X 的子节点集合记为 $C(X)$，我们称 X 为 X_1、.....、X_k 的父节点。根迭代是没有父节点的迭代，叶迭代则是没有任何子节点的迭代。某次迭代 X 的子树包含 X 及其所有后代节点，即经过迭代得到的子节点集合。

在我们的应用中，通常会遇到这样的特殊情况：如果 $|X| \geq 2$，迭代 X 不会直接访问任何元素，而是将 X 的所有元素传递给递归调用；如果 $|X| = 1$，则会访问 X 的唯一元素，且不会进行递归调用。也就是说，如果 $|X| > 1$，我们有 $X' = \emptyset$；如果 $|X| = 1$，则有 $X' = X$。

与次线性枚举中的惯例一样，我们假设每个被访问的元素都通过指向某个数据结构（通常是数组）的指针提供，并且在访问另一个元素之前，枚举过程会修改该数据结构。

2.2 势能分析与金字塔条件

本节将介绍一种势能分析方法，我们称之为 金字塔方法。

势能分析是一种成熟的技术，用于分析数据结构的摊还时间复杂度（例如，参见 [CLRS09]）。现在，我们定义一种变体，用于分析递归算法的摊还复杂度，尤其是组合枚举算法。其直觉是，对于许多枚举算法，被枚举集合中的大多数元素都在叶迭代中被访问。同时，这些迭代通常只需较少的计算时间。换言之，叶迭代通常能迅速连续地访问元素，且所需计算时间很少；而树的内部节点访问的元素较少，却需要更多计算时间。因此，叶迭代中使用的计算时间与产生的输出之间通常呈正向平衡，而内部节点则呈负向平衡。对于摊还时间分析，这提示我们建立这样一种模型：子节点将多余的时间资源传递给其父节点。Uno 的 Push-out 方法便采用了这一思路 [Uno 15], ，并据此证明了多种枚举算法具有常数延迟。

在我们的模型中，我们使用硬币 作为固定时间单位。每枚硬币可支付 $T^\star$ 条指令，其中 $T^\star$可以是输入规模的任意固定函数。最终， $T^\star$ 将是我们的

分析得出的摊还延迟。在我们的应用中，我们通常证明算法具有常数延迟，这相当于令 $T^\star = \mathcal{O}(1)$。如果一次迭代访问了一个元素，就会获得固定数量 $\mu > 0$ 的硬币作为奖励。执行指令时，算法需要支付相应数量的硬币。

按照上述直觉，我们的模型让子节点迭代将硬币传给父节点。这通过一个势函数 Φ来建模。它为每次迭代 X（即递归树中的每个节点）赋予一个势 $\Phi(X) > 0$，表示 X 传给其父节点的硬币数量。因此，一次迭代 X 会为访问的 μ 每个元素获得 X' 枚硬币，以及从其子节点获得 $\sum_{Y \in C(X)} \Phi(Y)$ 枚硬币，并将 $\Phi(X)$ 枚硬币传给其父节点（算法示例见图 2，对应示例 2.1）。

直观地说，如果以一种让每个节点都能支付自身计算时间的方式选择 Φ 和 $T^\star$，那么这一机制就是可持续的。也就是说，一次迭代 X 获得的 $\mu|X'| + \sum_{Y \in C(X)} \Phi(Y)$ 枚硬币与其传给父节点的 $\Phi(X)$ 枚硬币之差，必须足以支付执行 X 中指令的费用，即至少需要 $T(X)/T^\star$ 枚硬币。以下条件概括了这一点：

定义 2.2 (金字塔条件)。 若递归算法 $\mathcal{A}$ 满足势函数 Φ 和时间 $T^\star$ 下的金字塔条件，则对于 $\mathcal{A}$的每次迭代 X，都有

$$\sum_{Y \in C(X)} \Phi(Y) + \mu|X'| - \Phi(X) \geq \frac{T(X)}{T^\star}.$$

请注意，这要求根迭代 R 提供 Φ（R）枚硬币，以便传给它的（不存在的）父节点。这项额外的时间资源可用于支付递归之外执行的指令，例如读取输入或进行初始化。

我们现在证明，这个条件确实可用于为递归算法的摊还延迟提供界。

2.2.1 摊还延迟

引理 2.3。 设 $\mathcal{A}$ 是一个递归算法，用于枚举满足某个势函数 Φ 和时间 $T^\star$下金字塔条件的集合 X。设 X 是 $\mathcal{A}$的一次迭代。那么，在 X 的一棵子树中执行的指令数至多为 $T^\star(\mu|X| - \Phi(X))$。

证明。 我们通过对 X 的大小进行归纳法来证明该引理。在 X 的子树中执行的指令数量至多为

$$\begin{aligned}
&T(X) + \sum_{Y \in C(X)} T^\star(\mu|Y| - \Phi(Y)) \\
&= T^\star \left(\frac{T(X)}{T^\star} + \Phi(X) - \mu|X'| - \sum_{Y \in C(X)} \Phi(Y) \right) + T^\star(\mu|X| - \Phi(X)) \\
&\leq T^\star(\mu|X| - \Phi(X)).
\end{aligned}$$

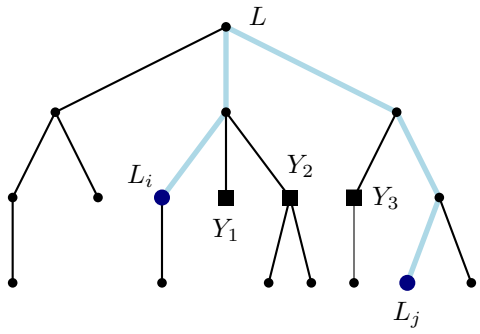


图 3：引理2.5的证明示意图。

注意，当 X 是叶节点，即 $C(X) = \emptyset$ 时，以及当 $C(X)$ 非空时，这个不等式都成立。在第二步中，我们用到了 $|X| = |X'| + \sum_{Y \in C(X)} |Y|$。最后一个不等式由金字塔条件得出。

□

由此立即得到以下结论：

定理 2.4。设 $\mathcal{A}$ 是一个枚举集合 X 的递归算法，并假设 $\mathcal{A}$ 对某个势函数 Φ 和时间 $T^\star$ 满足金字塔条件。枚举 X 的总时间为 $\mathcal{O}(|X|T^\star - \Phi(X))$。因此， $\mathcal{A}$ 的摊还延迟为 $\mathcal{O}(T^\star)$。

我们强调，这里只考虑递归过程中执行的计算，尤其不包括读取输入以及可能的初始化。

2.2.2 访问时间

枚举算法会为被枚举的集合 X 定义一种次序，即它访问集合 X 中各元素的顺序。现在我们分析按照此顺序访问第 i 个元素与第 j 个元素之间执行的指令数量。设 Δ 是递归树中从根节点到叶迭代的路径上所有迭代执行的指令数量的上界。

引理 2.5. 设 $\mathcal{A}$ 是一个枚举集合 X 的递归算法，并假设 $\mathcal{A}$ 满足金字塔条件，其时间为 $T^\star$。对于每个 $1 \leq i \leq j \leq |X|$，访问 X 中的第 i个元素与第 j个元素之间执行的指令数 δ_{ij}至多为 $2\Delta + (j-i)\mu T^\star - 1$。此外，在访问第 j个元素之前执行的指令数至多为 $\Delta + j\mu T^\star - 1$。

证明。 图3给出了一个示例。设 L_i 和 L_j 分别为访问第 i个和第 j个元素时的迭代。设 L 为 L_i 和 L_j在递归树中的最近公共祖先。设 $Y_1, \ldots, Y_k$ 表示在 L_i 和 L_j 之间执行的那些迭代，其父节点位于从 L 到 L_i 以及 L_j的路径上，但这些迭代本身不在这些路径上。

那么， δ_{ij} 至多等于这些路径上的指令数，加上 Y_1、……、Y_k的子树中的指令数。路径上的指令数至多为$2\Delta - T(L) \leq 2\Delta - 1$。根据引理2.3，在迭代 Y_i的子树中执行的指令数至多为 $\mu T^\star |Y_i|$。注意，集合 Y_1、……、Y_k包含了在 i和 j之间访问的所有元素，且每个元素恰好属于一个集合 Y_i。因此， $\sum_{i=1}^{k} \mu T^\star |Y_i| = \mu T^\star (j-i)$。由此得到 $\delta_{ij} \leq 2\Delta + \mu T^\star (j-i) - 1$。

第二部分的证明也可类似地得出：将从根节点到 L_j 的路径上执行的指令，以及所有根节点的父节点位于这条路径上且在 L_j 之前执行的子树中的指令相加（注意，第 j 个元素不包含在这些子树中的任何一个里）。□

这立即得出以下结果：

定理 2.6。设 $\mathcal{A}$ 是一个枚举集合 X 的递归算法，并假设 $\mathcal{A}$ 满足金字塔条件，其时间为 $T^\star$。那么，$\mathcal{A}$ 在时间 $\mathcal{O}(kT^\star + \Delta)$ 内枚举前 k 个元素。

注意，这给出了按照 Cappelli 和 Strozecki 的定义的摊还延迟上界 [CS23]。

2.2.3 与 Uno 的 Push-out 方法比较

Uno [Uno15] 提出了一种称为 *Push-out* 的方法，用于分析递归枚举算法的摊还复杂度。该方法使用以下条件：

定义 2.7（推出条件 [Uno15]）。一个递归算法满足推出条件，其时间为 $T^\star$，如果存在常数 $\alpha > 1$ 和 $\beta \geq 0$，使得对于所有迭代 X，都有

$$\sum_{Y \in C(X)} T(Y) \geq \alpha T(X) - \beta(|C(X)| + 1)T^\star.$$

定理 2.8（[Uno15，定理 1]）。满足时间为 $T^\star$ 的推出条件的递归枚举算法，其摊还延迟为 $\mathcal{O}(T^\star)$。

我们通过定义一个特定的势函数，证明金字塔方法包含推出方法；对于该势函数，两种方法给出的条件是等价的。

命题 2.9。若递归算法 $\mathcal{A}$ 满足 *Push-out* 条件，其时间为 $T^\star$，其中常数为 $\alpha > 1$ 和 $\beta \geq 0$，则 $\mathcal{A}$ 满足金字塔条件，其势函数对应的时间为 $T^\star$

$$\Phi(X) = \frac{T(X)}{(\alpha - 1)T^\star} + \frac{3\beta}{\alpha - 1}.$$

证明。我们有

$$\begin{aligned}\sum_{Y\in C(X)}\Phi(Y)-\Phi(X) &= \frac{1}{(\alpha-1)T^\star}\left(\sum_{Y\in C(X)}T(Y)-T(X)\right)+\frac{3\beta}{\alpha-1}(|C(X)|-1)\\ &\geq \frac{1}{(\alpha-1)T^\star}\left((\alpha-1)T(X)-\beta(|C(X)|+1)T^\star\right)+\frac{\beta}{\alpha-1}(3|C(X)|-3)\\ &= \frac{T(X)}{T^\star}+\frac{\beta}{\alpha-1}(2|C(X)|-4)\\ &\geq \frac{T(X)}{T^\star}.\end{aligned}$$

第一个不等式使用了 Push-out 条件，第二个不等式则由递归树中的每个内部节点至少有两个子节点这一事实得出。 □

这一结果结合定理2.4，得出了 [Uno15，定理 1]的另一种证明。此外，它表明，任何采用推出方法分析的枚举问题，也可以采用金字塔方法分析，并得到相同的摊还时间复杂度。特别地，我们可以利用引理2.5 来确定枚举前 k 个元素的复杂度。例如，这适用于枚举给定图中的消去序、匹配森林、连通诱导子图和生成树，这些问题由 [Uno15]研究。

在下文的例3.13 中，我们将说明，对于用于枚举偏序集中的理想的算法1.1，推出方法无法证明该算法具有常数延迟，而金字塔方法则可以。算法1.2也同样如此。

2.3 格雷码算法的最坏情况延迟

现在我们描述一种对递归算法进行去摊销化的方法 $\mathcal{A}$ ，该算法满足金字塔条件，且时间为 $T^\star$，即 $\mathcal{A}$具有摊还延迟 $T^\star$，前提是 $\mathcal{A}$按所谓的格雷码顺序枚举元素^。目标是得到一个算法 $\mathcal{A}$ ，其最坏情况延迟为 $\mathcal{O}(T^\star)$，并且按与 $\mathcal{A}$相同的顺序枚举元素。Uno [Uno03]曾在一份未发表的技术报告中描述过这种方法 [CS23]。为完整起见，我们在此介绍该方法。Capelli 和 Strozecki [CS23]提出了另一种去摊销化方法。

我们假设 $\mathcal{A}$ 按格雷码顺序枚举所有元素。这意味着，两个连续访问的元素之间对数据结构所做的修改可以用 $\mathcal{O}(1)$ 内存存储，并且可以在最坏情况下用 $\mathcal{O}(T^\star)$ 时间将数据结构修改为适用于下一个元素的状态。

$\mathcal{A}$ 的去摊还化使用一个包含 $\frac{2\Delta}{\mu T^\star}$ 个槽位的辅助队列 Q 。每个槽位存储两个连续访问的元素之间的所有变化。算法 $\mathcal{A}$ 按如下方式模拟算法 $\mathcal{A}$的计算：每当 $\mathcal{A}$ 访问一个元素时，算法 $\mathcal{A}$ 就将其存入队列 Q。更准确地说， $\mathcal{A}$ 保存 $\mathcal{A}$连续访问的两个元素之间数据结构中的变化，从而占用 Q的一个槽位。

首先，执行 $\mathcal{A}$ 的 Δ 条指令。随后，算法 $\mathcal{A}$每执行 $\mu T^\star$ 条指令，算法 $\mathcal{A}$ 就访问并移除 Q 中的第一个元素。具体而言， $\mathcal{A}$ 将 Q 第一个槽位中存储的所有变化应用到数据结构中。根据假设，这可以在 $\mathcal{O}(T^\star)$ 时间内完成。此外，每当 Q 包含的元素超过 $\frac{2\Delta}{\mu T^\star}$ 个时，算法$\mathcal{A}$就会访问并移除 Q中的第一个元素。下面需要证明：

引理 2.10。 每当算法 $\mathcal{A}$ 需要访问一个元素时，队列 Q 都非空。

证明。 假设 Q 在算法 $\mathcal{A}$ 需要访问第 j 个元素时为空。首先假设 Q 此前从未满过。当 $\mathcal{A}$ 需要访问第 j 个元素时，算法 $\mathcal{A}$已经执行了 $j\mu T^\star + \Delta$ 条指令。根据引理 2.5，算法 $\mathcal{A}$ 在访问第 j 个元素之前至多执行了 $j\mu T^\star + \Delta - 1$条指令。这与 Q为空的假设矛盾。

现在假设 Q 最后一次为满时，正是 $\mathcal{A}$访问第 i 个元素的时候。由队列的大小可知， $\mathcal{A}$ 执行了 $(j-i)\,\mu T^\star + 2\Delta$ 条指令。然而，根据引理 2.5，算法 $\mathcal{A}$ 在访问第 i 个元素和第 j 个元素之间至多执行了 $(j-i)\,\mu T^\star + 2\Delta - 1$ 条指令。这就产生了矛盾。 □

由此可得到以下关于去摊还的定理：

定理 2.11。 设 $\mathcal{A}$ 是一个满足金字塔条件、运行时间为 $T^\star$的递归算法。此外，假设 $\mathcal{A}$ 中连续访问的两个元素之间的变化可以用 $\mathcal{O}(1)$ 内存存储，且应用这些变化所需的时间为 $\mathcal{O}(T^\star)$。那么，算法 $\mathcal{A}$ 枚举与 $\mathcal{A}$相同顺序的所有元素，最坏情况延迟为 $\mathcal{O}(T^\star)$，同时需要 $\mathcal{O}(\Delta/T^\star)$额外空间，以及 $\mathcal{O}(\Delta)$ 初始化额外时间。

3 偏序集理想的常数摊还时间枚举

本节将介绍一个递归算法（算法 1.1），该算法以常数摊还延迟枚举输入偏序集 $(P, \preceq)$ 的所有理想。

3.1 预备知识

我们首先介绍一些辅助结构，它们将用于定义算法的递归结构。下文中，设 P 为一个偏序集，并固定其最长链 C 。设 $k = |C|$。枚举算法的递归过程将 P的所有理想划分为 $k+1$ 组，具体取决于理想中所包含的 C 的极大元素，或者理想是否不包含 C中的任何元素。

关于以下定义的示意图，请参见图 1。令 $\mathcal{I}(P)$ 表示偏序集 P 的理想集合。记 $C = \{c_1, \ldots, c_k\}$，其中 $c_1 \prec \cdots \prec c_k$。为简便起见，我们添加两个虚拟元素 c_0 和 c_{k+1}，它们分别小于和大于 P 中的所有元素。对于 $I \in \mathcal{I}(P)$，令 $\zeta(I) = \max\{i\colon\ c_i \in I\}$，并设 $\zeta(I) = 0$，如果 $I \cap C = \emptyset$。对于 $i \in \{0, \ldots, k\}$，令 $\mathcal{I}_i(P) = \{I \in \mathcal{I}(P)\colon\ \zeta(I) = i\}$。我们设 $D_i = \{u \in P\colon\ u \preceq c_i\}$ 为 c_i 的下集，并设 $U_i = \{u \in P\colon\ u \not\succeq c_{i+1}\}$。注意，$D_i$

而 U_i 和 $\mathcal{I}_i(P)$ 中最小和最大的理想分别是 $D_0 = \emptyset$ 和 $U_k = P$。特别地，我们有 $D_0 = \emptyset$ 和 $U_k = P$。

对于任意 $u \in P$，令

$$s_u = \max\{i \in \{0, \dots, k\} \colon c_i \preceq u\} \text{ and } l_u = \min\{i \in \{0, \dots, k\} \colon u \preceq c_i\}$$

分别表示 C 中小于 u 的最小元素的指标，以及 C 中大于 u的最大元素的指标。注意，$s_{c_i} = l_{c_i} = i$。对于 $i \in \{0, \dots, k+1\}$，令

$$S_i = \{u \in P \setminus C \colon s_u = i\} \text{ and } L_i = \{u \in P \setminus C \colon l_u = i\}.$$

注意，$L_1 = S_k = \emptyset$。此外，我们有 $D_i \setminus C = L_2 \cup \dots \cup L_i$和 $U_i \setminus C = S_1 \cup \dots \cup S_i$。令$P_i = (P \setminus D_i) \cap U_i$。换言之，$P_i$由 $u \in P$中满足 $u \not\preceq c_i$和 $u \not\succeq c_{i+1}$的那些元素组成。特别地，容易验证以下结论：

引理 3.1. 对于 $u \in P \setminus C$和 $i \in \{0, \dots, k\}$，以下性质等价：

1. $u \in P_i$,
2. $s_u \leq l_u$,
3. u 与 c_i 或 c_{i+1}不可比。

特别地，P_0 和 P_k 分别由 P 中与 c_1 以及与 c_k不可比的所有元素组成。由引理 3.1，容易得到以下结论：

引理 3.2。 $P \setminus C$的每个反链 A都至少包含于偏序集 P_0、.....、P_k中的一个。此外，A 至少属于两个这样的偏序集，当且仅当 C 包含一个与 A的所有元素都不可比的元素。

证明。取 $u \in A$ 使得 s_u 为极大。根据引理3.1，有$u \in P_{s_u}$。如果$A \not\subseteq P_{s_u}$，则存在 $v \in A \setminus P_{s_u}$。根据引理3.1 以及 u的极大性假设，这意味着 $l_v \leq s_u$。但这样一来，$v \preceq c_{l_v} \preceq c_{s_u} \preceq u$，这与 u 和 v 不可比相矛盾。第二个陈述可以类似地证明。 □

该引理意味着，$P \setminus C$中的每个元素至少属于偏序集 P_0、.....、P_k中的两个。特别地，L_i 中的每个元素都与 c_{i-1} 不可比，且 S_i 中的每个元素都与 c_{i+1}不可比。

现在我们将 P的理想集合分解。显然，$\mathcal{I}(P) = \mathcal{I}_0(P) \dot\cup \dots \dot\cup \mathcal{I}_k(P)$。下一个引理建立了 $\mathcal{I}_i(P)$与 $\mathcal{I}(P_i)$之间的一一对应关系，这是递归的核心要素。

引理 3.3。 对于 $i \in \{0, \dots, k\}$，映射

$$\varphi_i^P \colon \mathcal{I}_i(P) \to \mathcal{I}(P_i),\ I \mapsto I \setminus D_i$$

是一个（定义良好的）双射。其逆映射将 $I' \in \mathcal{I}\ (P_i)$ 映射为 $I' \cup D_i \in \mathcal{I}_i\ (P)$ 。

证明。设 $I \in \mathcal{I}_i(P)$。由于 I 向下封闭，因此有 $D_i \subseteq I$。此外，由于 $c_{i+1} \notin I$，有$I \cap (S_{i+1} \cup \cdots \cup S_{k-1}) = \emptyset$。因此，$I \setminus D_i$ 是 P_i的一个理想。现在设 $I' \in \mathcal{I}(P_i)$。显然，$I' \cup D_i$向下封闭，因此是 P 的一个理想。

对于 $I \in \mathcal{I}_i(P)$，我们有 $(I \setminus D_i) \cup D_i = I$如 $D_i \subseteq I$。反之，对于 $I' \in \mathcal{I}(P_i)$，我们有 $(I' \cup D_i) \setminus D_i = I'$如 $I' \cap D_i \subseteq P_i \cap D_i = \emptyset$。特别地，$\varphi_i^P$是一个双射。 □

我们以若干观察结果结束本部分，这些结果将在第 3.3 节中用于为 $\mathcal{I}(P)$定义一个格雷码。

引理 3.4。设 $i \in \{0, \ldots, k-2\}$。

1. 对于 $I \in \mathcal{I}_i(P)$，集合 $I \cup \{c_{i+1}, c_{i+2}\}$是 P的一个理想，当且仅当 $L_{i+1} \cup L_{i+2} \subseteq I$。因此，$U_i \cup \{c_{i+1}, c_{i+2}\} \in \mathcal{I}_{i+2}(P)$和 $U_i \setminus D_{i+2} \in \mathcal{I}(P_{i+2})$。*2.* 类似地，对于 $I \in \mathcal{I}_{i+2}(P)$，集合 $I \setminus \{c_{i+1}, c_{i+2}\}$是 P的一个理想，当且仅当 $(S_{i+1} \cup S_{i+2}) \cap I = \emptyset$。因此，$D_{i+2} \setminus \{c_{i+1}, c_{i+2}\} \in \mathcal{I}_i(P)$和 $D_{i+2} \setminus (D_i \cup \{c_{i+1}, c_{i+2}\}) \in \mathcal{I}(P_i)$。

证明。我们只证明第一个结论，第二个结论的证明类似。令 $I \in \mathcal{I}_i(P)$。首先，假设$I \cup \{c_{i+1}, c_{i+2}\}$是 P的理想。那么 $L_{i+1} \cup L_{i+2} \subseteq D_{i+1} \cup D_{i+2} \subseteq I \cup \{c_{i+1}, c_{i+2}\}$，因为该集合是向下封闭的。反过来，假设 $L_{i+1} \cup L_{i+2} \subseteq I$。我们有 $D_{i+2} = D_i \cup L_{i+1} \cup L_{i+2} \subseteq I$，因为$D_i \subseteq I$由 $I \in \mathcal{I}_i(P)$ 得出，因此 $I \cup \{c_{i+1}, c_{i+2}\}$是向下封闭的。特别地，这就得到$U_i \cup \{c_{i+1}, c_{i+2}\} \in \mathcal{I}_{i+2}(P)$。由引理 3.3，我们进而有 $\varphi_{i+2}^P(U_i \cup \{c_{i+1}, c_{i+2}\}) = U_i \setminus D_{i+2} \in \mathcal{I}(P_{i+2})$。 □

3.2 常数摊还时间算法

以下设 P 是一个包含 n个元素的偏序集。现在，我们描述一种算法，它能以常数摊还延迟列出 P 的所有理想。本节将介绍一个基本版本（算法1.1），它不一定按格雷码顺序列出理想。在下一节中，我们将说明如何通过为递归调用选择特定顺序来修改此版本，使其按格雷码顺序输出。

我们采用第3.1节中的记号。此外，令 Q表示 P 的不可比对集合，并令 $q = |Q|$。

算法说明我们使用 $\mathcal{I}(P)$ 与 $\mathcal{I}(P_0) \dot{\cup} \cdots \dot{\cup} \mathcal{I}(P_k)$ 之间的双射，该双射在引理 3.3中进行了描述。给定一个输入偏序集 P，算法 1.1 的一次迭代包括计算偏序集 P_0、.....、 P_k，并递归处理这些更小的偏序集。在整个递归过程中，维护一个元素列表 I，其中包含父节点迭代已添加到当前处理理想中的元素。根据引理 3.3，通过将 D_i中的元素添加到 P_i 的一个理想中，我们可以得到 P 的一个理想。因此，在针对偏序集 P_i的递归调用开始时，会将集合 $D_i = D_{i-1} \cup L_i \cup \{c_i\}$ 添加到数组 I 中。

实现现在我们将详细说明算法 1.1 各步骤的实现以及所需的数据结构。

输入与输出：输入偏序集 P 由邻接矩阵 M 表示，其中 $M_{uv} = 1$ 若$u \prec v$ 且 $M_{uv} = 0$，否则。在初始化阶段，我们重新标记 P 中的元素，以保证 $u \preceq v$ 蕴含 $u \leq v$。算法1.1 访问到的理想存储在数组 I中。

最长链的计算。现在我们描述最长链的计算过程（第5行）。为了使算法1.1 以常数摊还延迟运行，这一计算的时间复杂度需要为 $\mathcal{O}(n + q)$。特别地，我们不能使用拓扑排序算法，因为它需要$\Omega(n^2)$ 时间。

相反，我们首先使用以下算法计算 P的反链分解（参见算法3.1）：我们按顺序 $\mathbf{1}$ P逐个处理，... ...，n。整个过程中，

算法 3.1： 创建反链分解。

```
1  过程 主程序(偏序集 P由元素 {1,...,n}构成，使得 u ⪯ v蕴含 u ≤ v)
2      j ← 0
3      对于 i ← 1,...,n do
4          对于 t ← j,...,1执行
5              对于 v ∈ A_t 执行
6                  如果 v ≺ u 则
7                      如果 t = j，则
8                          A_{j+1} ← {u} // 创建一个新的反链 A_{j+1}
9                          x_u = v
10                         j ← j + 1
11                         转到 迭代结束 // 跳出两层内循环
12                     else
13                         A_{j+1} ← A_{j+1} ∪ {u} // 将元素添加到现有反链中
                               A_{j+1}
14                         x_u = v
15                         转到迭代结束
16             A_1 ← A_1 ∪ {u} // 未找到更小的元素，因此将 u 添加到底部
                   反链 A_1
17             x_u = ⊥
18         迭代结束:
19     返回 A_1,...,A_j
```

我们维护一个反链列表 $A_1, \ldots, A_j$，并为每个先前处理过的元素 u维护一个变量 x_u。插入新元素 u 时，依次将它与反链$A_j, \ldots, A_1$ 中的元素进行比较，直到找到第一个满足 $v \prec u$ 的元素 $v \in A_i$。此时我们设置 $x_u = v$，并将 u

添加到 A_{i+1}。这可能会导致创建一个新的反链 A_{j+1}。如果 u与所有先前处理过的元素都不可比，则将 u 添加到 A_1，并将 $x_u = \perp$ 设为空指针。由此得到 P 的反链分解 $A_1, \ldots, A_k$。

然后，我们从任意元素 $u \in A_k$ 出发，沿着变量 x_u，找到偏序集 P 中的一条最长链 C。显然，这会构造出一条大小为 k 的链 C。由于每条链在每个反链 A_1、.....、A_k 中至多包含一个元素，因此 C 是偏序集 P 中的一条最长链。

该算法的正确性由以下引理得出：

引理 3.5. 对于每个 $i \in \{1, \ldots, k\}$，由算法 A_i 计算得到的集合 A_i 算法 *3.1* 是一个反链。算法 *3.1* 以及最长链的计算，都可以在 $\mathcal{O}(n+q)$ 时间内完成。

证明。 假设存在 $i \in \{1, \ldots, k\}$ 和 u，$v \in A_i$ 满足 $u \prec v$。那么 $u \leq v$，因此 v 是在 u 之后插入的。于是 v 本应与 u 进行比较，进而被插入 A_{i+1} 中，这就产生了矛盾。

算法 3.1 将每个元素与至多一个较小元素进行比较。所有其他比较都是在不可比元素之间进行的，并且每一对不可比元素至多测试一次。因此，算法 3.1的时间复杂度，也就是构造最长链的时间复杂度，为 $\mathcal{O}(n+q)$。 □

计算 s_u 和 l_u。为了计算所有 $u \in P \setminus C$ 对应的 s_u 和 l_u（算法 1.1 第 6 行），我们使用之前计算出的反链分解 $A_1, \ldots, A_k$。令 $s_{c_i} = i$，其中 $i = 1, \ldots, k$。令 $u \in P \setminus C$，并令 A_i 为包含 u的反链。根据 C的构造，我们有 $c_i \in A_i$，因此 $s_u < i$。于是，我们将 u与 $c_{i-1}, \ldots, c_1$ 进行比较，直到找到 C 上小于 u 的极大元素。l_u的值以类似方式确定。如引理 3.5 所示，该过程的时间复杂度为 $\mathcal{O}(n+q)$。

对列表排序。通过在预处理步骤中重新标记 P 的元素，我们确保 $u \preceq v$ 蕴含 $u \leq v$。在递归调用中，我们需要保持这一性质。在更新过程中，我们用 $(P' \setminus L_i) \cup S_i$替换 P'。为了从 P'中移除 L_i，我们遍历 P'，移除所有满足 $s_u = i$的顶点 u。我们可以将 S_i 构造成有序列表，因此将 S_i 添加到 P' 相当于合并两个有序列表。创建有序列表 $P_1, \ldots, P_k$的时间复杂度为 $\mathcal{O}(\sum n_i)$，其中 $n_i = |P_i|$。引理 3.1 意味着 $\sum n_i \leq q + n - k \in \mathcal{O}(n+q)$。

复杂度分析 根据上一段中描述的设置，我们得到初始化和单次迭代的如下时间复杂度：

引理 3.6. 算法 *1.1*的初始化需要 $\mathcal{O}(n^2)$ 时间。一次迭代可在 $\mathcal{O}(n+q)$内完成。最大递归深度为 n。特别地，递归树中从根节点到叶节点的路径上所有迭代的时间复杂度为 $\mathcal{O}(n(n+q))$。

证明。 在初始化过程中，算法 1.1 以一种使 P 中的元素重新编号的方式，使得 $u \preceq v$ 蕴含$u \leq v$对所有 u，$v \in P$。这可以通过经典的拓扑排序算法在 $\mathcal{O}(n^2)$ 时间内完成。我们展示了如何计算最长链，并构造集合 L_i 和 S_i（第 5 行和第 7 行），耗时 $\mathcal{O}(n+q)$。更新包含 P' 和 I 的数组（第 10 行和第 11 行）总共需要

时间 $\mathcal{O}(n+q)$，因为每个元素最多被添加和移除一次。因此，递归函数的一次迭代的运行时间为 $\mathcal{O}(n+q)$。由于 P_i 不包含 C中的任何元素，递归深度至多为 n。

□

关于算法 1.1 的空间需求，我们注意到以下结论：

引理 3.7。 算法 *1.1*的空间复杂度为 $\mathcal{O}(n^2)$。

证明。 算法 1.1 的每次迭代都需要 $\mathcal{O}(n)$ 内存来存储一条最长链、子偏序集 P'、数值 s_u 和 l_u 以及集合 S_i 和 L_i。这些变量必须存储在调用栈上，其高度至多为 n，由引理3.6可知。

□

现在我们转向势能分析。在定义相关概念之前，我们先证明以下数论引理。

引理 3.8。 对于任意 $a_1, \ldots, a_g, b_1, \ldots, b_f \in \mathbb{Z}_{>0}$，有

$$48\sum_{j=1}^{g} j\left(\binom{a_j}{3}+1\right)+48\sum_{j=1}^{f} j\left(\binom{b_j}{3}+1\right)\geq \sum_{j=1}^{g} a_j \cdot \sum_{j=1}^{f} b_j.$$

证明。 由詹森不等式可得

$$\frac{\sum_{j=1}^{g} \frac{1}{\sqrt{j}}\left(\sqrt{j}a_j\right)^3}{\sum_{j=1}^{g} \frac{1}{\sqrt{j}}} \geq \left(\frac{\sum_{j=1}^{g} \frac{1}{\sqrt{j}}\sqrt{j}a_j}{\sum_{j=1}^{g} \frac{1}{\sqrt{j}}}\right)^3,$$

这意味着

$$\sum_{j=1}^{g} j a_j^3 \geq \frac{\sum_{j=1}^{g} a_j}{\left(\sum_{j=1}^{g} \frac{1}{\sqrt{j}}\right)^2}\left(\sum_{j=1}^{g} a_j\right)^2 \geq \frac{1}{4}\left(\sum_{j=1}^{g} a_j\right)^2$$

其中最后一个不等式由 $\sum_{j=1}^{g} a_j \geq g$和 $\sum_{j=1}^{g} \frac{1}{\sqrt{j}} 2\sqrt{g}$ 得出。由 $2(a_j-1)(a_j-2)+12 \geq a_j^2$，可得

$$12\sum_{j=1}^{g} j\left(\binom{a_j}{3}+1\right)\geq \sum_{j=1}^{g} j a_j^3.$$

不失一般性，我们假设 $\sum_{j=1}^{g} a_j \geq \sum_{j=1}^{f} b_j$。结合前面的不等式，可得

$$48\sum_{j=1}^{g} j\left(\binom{a_j}{3}+1\right)+48\sum_{j=1}^{f} j\left(\binom{b_j}{3}+1\right)\geq 4\sum_{j=1}^{g} j a_j^3 \geq \left(\sum_{j=1}^{g} a_j\right)^2 \geq \sum_{j=1}^{g} a_j \cdot \sum_{j=1}^{f} b_j. \quad \square$$

我们的目标是利用上一节的金字塔条件（见定义 2.2），证明算法 1.1 具有摊销常数延迟。为此，我们定义如下划分 $Q = Q' \dot\cup Q'' \dot\cup Q'''$：令 Q' 为其中一个顶点属于 C 的不可比对集合，令 Q'' 为不可比对 $\{u, v\}$ 的集合，其中 u、$v \in P \setminus C$ 满足 u 和 v 在 C 上有一个共同的不可比元素；令 Q''' 为 Q 中剩余的对。回顾一下，$n_i = |P_i|$。令 Q_i 为 P_i 中不可比对的集合，并设 $q_i = |Q_i|$。设 $Q''_i = Q'' \cap Q_i$ 和 $q''_i = |Q''_i|$。类似地，我们定义 Q'''_i 和 q'''_i。令 T 为 P 中所有 3-反链的集合。如上所述，我们定义划分 $T = T' \dot\cup T'' \dot\cup T'''$，并令 t'、t''、t''' 分别表示 T'、T''、T''' 的大小。对于 $i = 1, \dots, k$，令 $T_i \subseteq T$ 表示包含在 P_i 中的 3-反链，并设 $t_i = |T_i|$。设 $T''_i = T'' \cap T_i$ 和 $T'''_i = T''' \cap T_i$，并定义$t''_i = |T''_i|$ 和 $t'''_i = |T'''_i|$。

对于势能分析，我们需要关于 q''' 的以下上界。

引理 3.9. 有 $q''' \le 96(\sum t''_i + \sum n_i)$。

证明. 设 $\{u, v\} \in Q'''$。由于 u 和 v 不可比，我们有 $s_u < l_v$ 和 $s_v < l_u$，因此 $\max\{s_u, s_v\} < \min\{l_u, l_v\}$。若对某个 $i \in \{1, \dots, k\}$，有 $\max\{s_u, s_v\} < i < \min\{l_u, l_v\}$，则 c_i与 u 和 v均不可比。这与假设 $\{u, v\} \in Q'''$矛盾。因此，我们有 $\max\{s_u, s_v\} = \min\{l_u, l_v\} - 1 =: i$。由于 $l_u \ge s_u + 2$，可得 $s_u = l_v - 1$或 $s_v = l_u - 1$。这给出了一个划分 $Q''' = G_1 \dot\cup \cdots \dot\cup G_k$，其中 G_i由所有满足 $u \in S_i$ 和 $v \in L_{i+1}$的对 $u, v \in Q'''$组成。显然，我们有 $|G_i| \le |S_i||L_{i+1}|$。

现在，我们将由 S_i 诱导的 P 子偏序集分解为反链。令$A_{i,}$1为 S_i的极小元素集合。令$A_{i,}$2为 $S_i \setminus A_1$的所有极小元素。我们以此类推，直到到达最后一个非空反链，记为A_{i,e_i}。类似地，令$B_{i,}$1表示 L_{i+1}的极大元素集合，令$B_{i,}$2表示$L_{i+1} \setminus B_{i,}$1的极大元素集合，并继续下去，直到最后一个非空反链B_{i,f_i}。令$a_{i,j} = |A_{i,j}|$ 和$b_{i,j} = |B_{i,j}|$。利用引理 3.8，我们得到

$$|G_i| \le |S_i||L_{i+1}| \le \sum_{j=1}^{e_i} a_{i,j} \cdot \sum_{j=1}^{f_i} b_{i,j} \le 48 \sum_{j=1}^{e_i} j \left(\binom{a_{i,j}}{3} + 1 \right) + 48 \sum_{j=1}^{f_i} j \left(\binom{b_{i,j}}{3} + 1 \right).$$

令 Z 为包含 C中一个元素的4-反链集合，令 Z_i 为 Z 中包含 S_i的三个元素的4-反链集合。由于 C 是 P的一条极大链，$A_{i,j}$中的所有元素都与 $c_{i+1}, \dots, c_{i+j}$不可比。因此，Z_i 包含$j\binom{a_{i,j}}{3}$ 4-反链，其中包含来自$A_{i,j}$的三个元素。这意味着

$$\sum_{i=1}^{k-1} \sum_{j=1}^{e_i} j \binom{a_{i,j}}{3} \le \sum_{i=1}^{k-1} |Z_i| \le |Z| = \sum t''_i.$$

由于每个反链 $A_{i,\ j}$都非空，且 $A_{i,\ j} \subseteq P_{i+1} \cap \cdots \cap P_{i+j}$，因此我们得到

$$\sum_{i=1}^{k-1} \sum_{j=1}^{e_i} j \le \sum n_i.$$

对 $B_{i,\ j}$ 使用类似的界，我们即可得到所需的不等式。 □

我们令 $\alpha = 922$, $\beta = 921$, $\gamma = 385$, $\delta = 192$ 并定义势函数 $\Phi(P) = \alpha + \beta n + \gamma q + \delta t$。利用该势函数和金字塔条件（见定义 2.2），我们现在证明算法1.1 具有常数摊还延迟。

引理 3.10。对于每个非空偏序集 P，有 $\sum \Phi(P_i) - \Phi(P) \geq n + q$。特别地，算法 *1.1* 具有常数摊还延迟。

证明。我们证明 $\sum \Phi(P_i) - \Phi(P) - (n+q) \geq 0$。首先，我们分析此表达式中对应于偏序集元素的各项。有 $n \leq k + \frac{1}{2}\sum n_i$ 根据引理 3.1 可得 $2(n-k)_{\perp} \leq \sum n_i$。因此，

$$\beta \sum n_i - \beta(n+1) \geq \frac{\beta - 1}{2} \sum n_i - (\beta + 1)k.$$

其次，我们分析不可比对的贡献。对于每个形如 $\{c_i, v\}$ 且满足 $v \in P \setminus C$的不可比对，有 $v \in P_i$。引理 3.2 给出 $q' + n - k = \sum n_i$，因此特别有 $q' \leq \sum n_i$。有 $q'' \leq \frac{1}{2}\sum q_i''$根据引理 3.2，集合 Q'' 中的任意一对元素都至少属于以下偏序集中的两个 P_0、.....、P_k。最后，引理 3.2 给出 $q''' = \sum q_i'''$。此外，根据引理 $q''' \leq 96(\sum t_i'' + \sum n_i)$ 3.9，有。结合这些不等式可得

$$\gamma \sum q_i - (\gamma + 1)q \geq \frac{\gamma - 1}{2} \sum q_i'' - (\gamma + 97) \sum n_i - 96 \sum t_i''.$$

最后，我们分析与 3-反链对应的项。对于形如 $t' = \sum q_i''$ 且满足 $\{c_i, u, v\}$ 的每个 3-反链，其中 u, $v \in P \setminus C$，有 $\{u, v\} \in Q_i''$。由引理 $t'' \leq \frac{1}{2}\sum t_i''$ 可知，3.2每个 3-反链 T'' 至少属于 P_0, ..., P_k 中的两个偏序集。最后，我们有$t''' = \sum t_i'''$，这同样由引理 3.2 得出。综上可得

$$\delta \left(\sum t_i - t \right) \geq \frac{\delta}{2} \sum t_i'' - \delta \sum q_i''.$$

合并以上不等式，我们得到

$$\begin{aligned}
&\sum \Phi(P_i) - \Phi(P) - n - q \\
&\geq \alpha k + \frac{\beta - 1}{2} \sum n_i - (\beta + 1)k \\
&\quad + \frac{\gamma - 1}{2} \sum q_i'' - (\gamma + 97) \sum n_i - 96 \sum t_i'' \\
&\quad + \frac{\delta}{2} \sum t_i'' - \delta \sum q_i'' \\
&\geq k(\alpha - \beta - 1) + \sum n_i \left(\frac{\beta - 1}{2} - \gamma - 97 \right) + \sum q_i'' \left(\frac{\gamma - 1}{2} - \delta \right) + \sum t_i'' \left(\frac{\delta}{2} - 96 \right) \\
&= 0.
\end{aligned}$$

根据引理 3.6，算法 1.1 的一次迭代可在时间 $\mathcal{O}(n+q$ 内执行。由上述内容可知，势函数 Φ 满足关于 $T^\star = \mathcal{O}(1)$ 的金字塔条件。根据定理 2.4，算法 1.1 具有常数摊还延迟。

□

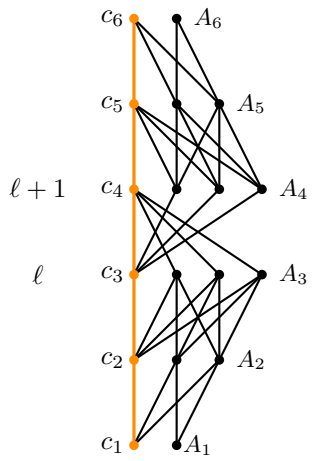


图 4：偏序集的构造 P，对于该偏序集，算法 1.1 无法使用推出方法进行分析。图中使用了 $\ell = 3$。

下面的定理总结了本节的结果：

定理 3.11。 算法 *1.1* 以常数摊还延迟枚举所有理想，且所需$\mathcal{O}\big(n^2\big)$ 空间。对于 $k \in \mathbb{N}$，第 k 个理想在时间 $\mathcal{O}(k + n(n + q))$ 内被访问。

备注 3.12。 无需要求输入偏序集以其邻接矩阵的形式给出，也可以修改算法 *1.1*，使输入偏序集以其哈斯图的形式给出。思路如下：设 m 为哈斯图中的边数。我们证明，单次迭代需要 $\mathcal{O}($ $n + m)$ 条指令。进行势能分析时，我们采用形式为 $\Phi(P) = \alpha + \beta n + \gamma m + \delta q$的势能。由此，与之前一样，可以证明该算法实现了常数摊还延迟。此处省略细节。

我们通过说明无法使用推出方法（见定理2.8）来证明算法 1.1 具有常数摊还延迟，从而结束本节。

例 3.13。 设 $\ell \in \mathbb{N}$。我们构造一个偏序集 P，使得推出方法（见定理 *2.8*）无法证明算法 *1.1* 具有常数摊还延迟。图 *3.13* 给出了示意图。

我们从 2ℓ 个反链的不交并开始，这些反链为 A_1、……、 $A_{2\ell}$，其中 $A_{\ell+1-i}$和 $A_{\ell+i}$的大小为 $\lceil 1+\ell/i \rceil$ （取整不会改变后续计算中的渐近结果）。我们按如下方式添加覆盖关系：固定 $c_i \in A_i$，其中 $i = 1$、……、 2ℓ，并令 $c_i \prec c_{i+1}$，其中 $i \in \{1, \ldots, 2\ell - 1\}$。因此， $\{c_1, \ldots, c_{2\ell}\}$ 是 P中的一条最长链。对于 $i \in \{2, \ldots, \ell\}$、 $a \in A_i \setminus \{c_i\}$ 和 $b \in A_{i-1}$，令 $b \prec a$。对于所有 $a \in A_\ell$，令 $a \prec c_{\ell+1}$。对称地，对于所有 $i \in \{\ell + 1, 2\ell - 1\}$、 $a \in A_i \setminus \{c_i\}$ 和 $b \in A_{i+1}$，令 $a \prec b$；对于所有 $a \in A_{\ell+1}$，令 $c_\ell \prec a$ 。

我们有 $n = 2\sum_{i=1}^{\ell} \lceil 1 + \ell/i \rceil = \Theta(\ell$ 对数 $\ell)$。 P中的不可比对具有以下形式：来自同一反链的元素对，以及 $(A_i \setminus \{c_i\}) \times (A_j \setminus \{c_j\})$ 中的所有元素，对应于

$i \le \ell < j$，以及 $\{c_i, a\}$ 与 $a \in A_j \setminus \{c_j\}$ 和 $j < i$ 形式的配对（如果 $i \le \ell - 1$），以及 $j > i$（如果 $i \ge \ell + 1$）。因此 $q = \Theta(n^2) = \Theta(\ell^2 \log^2 \ell)$。对于 $i \le \ell - 1$，我们有 $P_i = \bigcup_{j=1}^{i+1} A_i \setminus \{c_i\}$。因此$n_i = \Theta(\ell \log i)$ 且 $q_i = \Theta(\sum_{j=\ell-i}^{\ell} (\ell/j)^2)$。$i \ge \ell + 1$也有类似的公式。最后，我们有$P_\ell = P \setminus C$，因此 $n_\ell \le n$ 和 $q_\ell \le q$。

回顾一下，算法 *1.1* 处理 P的迭代复杂度为 $T(P) = \Theta(n + q) = \Theta(\ell^2 \log^2 \ell)$。除 P_ℓ之外，处理 P的所有子节点迭代的时间复杂度为

$$\sum_{i=0}^{2\ell} T(P_i) - T(P_\ell) = \Theta\left(\sum_{i=1}^{\ell-1} \sum_{j=\ell-i}^{\ell} (\ell/j)^2\right) = \Theta(\ell^2 \log \ell)$$

因为

$$\sum_{i=1}^{\ell-1} \sum_{j=\ell-i}^{\ell} \frac{1}{j^2} = \sum_{i=1}^{\ell} i \frac{1}{i^2} - \frac{1}{\ell^2} = \Theta(\log \ell).$$

此外，$(|C(P)| + 1)T^\star = (2\ell + 1)T^\star = \Theta(\ell)$。现在假设算法 *1.1* 对某些 $\alpha > 1$ 和 $\beta \ge 0$满足 ***Push-out*** 条件。特别地，这意味着

$$\sum_{i=0}^{2\ell} T(P_i) \ge \alpha T(P) + \beta(2\ell + 1)T^\star.$$

由此得到

$$\begin{aligned}
\Theta(\ell^2 \log \ell) &= \sum_{i=0}^{2\ell} T(P_i) - T(P_\ell) \\
&\ge \alpha T(P) - T(P_\ell) - \beta(2\ell + 1)T^\star \\
&\ge (\alpha - 1)T(P) - \beta(2\ell + 1)T^\star \\
&= \Theta(\ell^2 \log^2 \ell)
\end{aligned}$$

（注意 $\alpha - 1 > 0$）。通过选择 $\ell \in \mathbb{N}$ 足够大，我们便得到矛盾。因此，算法 *1.1* 不满足具有常数时间的 ***Push-out*** 条件。

3.3 偏序集理想的格雷码枚举

本节将修改算法 1.1 ，以格雷码顺序列出输入偏序集的理想 P，同时保持常数摊还延迟。具体做法是为算法 1.1 中的递归调用指定一种特定顺序，并为每个递归调用指定起始理想和终止理想。

对于 $k \in \mathbb{N}$，令 $\mathcal{F}^k(P$为如下翻转图：其顶点是 P的理想，若两个理想相差的元素至多为 k 个，则在它们之间连一条边。Pruesse and Ruskey [PR93]证明了 $\mathcal{F}^2(P)$ 存在一条哈密顿路径。注意，要获得一种枚举理想的无循环算法，只需在任意常数 $\mathcal{F}^m(P$下的 $m)$ 中找到一条可高效计算的哈密顿路径。

我们首先证明，可以选择算法 1.1 中递归调用的顺序，使得连续访问的两个理想相差的元素至多为三个。我们分两步进行：首先证明，在 $\mathcal{F}^3(P$中存在一条哈密顿路径，它按 P_i的顺序连续列出各理想，并在归纳步骤中保持这一性质。随后，我们讨论如何将其转换为算法。我们采用第 3.1 节中的记号。

引理 3.14。对于每个理想 $I \neq P$，翻转图 $\mathcal{F}^3(P)$ 中都存在一条连接 I和 P的哈密顿路径。类似地，对于每个理想 $I \neq \emptyset$，翻转图 $\mathcal{F}^3(P)$ 中都存在一条连接 I和 $\emptyset$的哈密顿路径。

证明。我们通过对 $|P|$ 使用归纳法，并行证明这两个结论。容易看出，当 P 是链时，结论成立。因此，以下我们假设 P 包含不可比较元素。首先假设 $I \neq P$，并在$\mathcal{F}^3(P)$中构造一条从 I 到 P 的哈密顿路径。设 C为 P中的一条最长链，并令 $\zeta = \zeta(I)$。我们分两种情况讨论：

情况 1： $0 \leq \zeta < k$。我们区分 ζ 为偶数或奇数的情况。

首先假设 ζ 为偶数。回顾引理 $\mathcal{I}_i(P)$ 与 $\mathcal{I}(P_i)$ 之间由引理 3.3给出的对应关系。利用这一对应关系，我们列出 P的理想，按照递归顺序处理偏序集 $P_0, \ldots, P_k$，顺序为

$$P_\zeta, P_{\zeta-2}, \ldots, P_0, P_1, P_3, \ldots, P_{\zeta-1}, P_{\zeta+1}, P_{\zeta+2}, \ldots, P_k. \tag{1}$$

对于每个 $i \in \{0, \ldots, k\}$，我们定义 P的理想 I_i^f和 I_i^l，使得 $\mathcal{I}(P_i)$ 中最先和最后访问的理想分别为 $\varphi_i^P(I_i^f)$ 和 $\varphi_i^P(I_i^l)$。我们区分多种情况。对于 $\zeta > 0$，情况如表 1所示。注意，根据引理 3.4，列出的所有集合都是 P 的理想。我们分别用 $P_i^{\min}$ 和 $P_{\max i}$ 表示 P_i 中的极小元素集合和极大元素集合。对于 $\zeta = 0$，顺序为 P_0，P_1，……，P_k。我们使用

$$I_0^f = I \text{ and } I_0^l = \begin{cases} U_0 & \text{if } I \neq U_0 \text{ or } P_0 = \emptyset \\ U_0 \setminus \{x\} & \text{otherwise, where } x \in P_0^{\max}, \end{cases}$$

以及

$$I_1^f = \begin{cases} U_0 \cup \{c_1\} & \text{if} \neq I_1^l \text{ or } P_1 = \emptyset \\ U_0 \cup \{c_1\} \setminus \{x\} & \text{else, where } x \in P_1^{\max} \end{cases} \text{ and } I_1^l = U_1$$

（因此表 1中对应于 0 和 1 的行，只需将 $I_0^f = I$改为）。对于 $i \geq 2$，理想根据表 $i > \zeta + 1$ 中的情况 1定义。

现在令 $\zeta \in \{0, \ldots, k-1\}$。我们通过归纳法证明，$\mathcal{F}^3(P)$ 包含所需的哈密顿路径。对于每个 $i \in \{0, \ldots, k\}$，I_i^f和 I_i^l中必有一个等于 D_i或 U_i，因此它在 φ_i^P下的像是 $\emptyset$ 或 P_i。根据构造，当且仅当 $P_i = \emptyset$时，$I_i^f = I_i^l$成立；此时 P_i 只有一个理想。由表 1 可轻易验证，只要 P_i、P_j 在式 (1) 给出的序列中相邻出现，那么 I_i^l和 I_j^f 至多相差三个元素。通过归纳法，我们得到一条哈密顿路径

i	I_i^f	I_i^l
ζ，如果 $\zeta > 0$	I	$\begin{cases} D_\zeta & \text{如果 } \neq I \text{ 或 } P_\zeta = \emptyset \\ D_\zeta \cup \{x\} & \text{否则，使用 } x \in P_\zeta^{\min} \end{cases}$
$2 \le i \le \zeta - 2$，$i \equiv \zeta$（模 2）	$D_{i+2} \setminus \{c_{i+1}, c_{i+2}\}$	$\begin{cases} D_i & \text{如果 } \neq I_i^f \text{ 或 } P_i = \emptyset \\ D_i \cup \{x\} & \text{否则，满足 } x \in P_i^{\min} \end{cases}$
0	——〃——	$\begin{cases} U_0 \text{ 如果} & \neq I_0^f \quad P_0 = \emptyset, \\ U_0 \setminus \{x\} & \\ \text{否则满足 } x \in P_0^{\max} & \end{cases}$
1	$\begin{cases} U_0 \cup \{c_1\} & \text{如果 } \neq U_1 \text{ 或 } P_1 = \emptyset \\ U_0 \cup \{c_1\} \setminus \{x\} & \text{否则，满足 } x \in P_1^{\max} \end{cases}$	U_1
$3 \le i \le \zeta - 1$，$i \not\equiv \zeta$ 2（模）	$\begin{cases} U_{i-2} \cup \{c_i, c_{i-1}\} & \text{如果 } \neq U_i \text{ 或 } P_i = \emptyset \\ U_{i-2} \cup \{c_i, c_{i-1}\} \setminus \{x\} & \text{否则，取 } x \in P_i^{\max} \end{cases}$	U_i
$\zeta + 1$，若 $\zeta > 0$	——〃——	——〃——
$\zeta + 2 \le i \le k$	$\begin{cases} U_{i-1} \cup \{c_i\} & \text{若 } \neq U_i \text{ 或 } P_i = \emptyset \\ U_{i-1} \cup \{c_i\} \setminus \{x\} & \text{否则，取 } x \in P_i^{\max} \end{cases}$	——〃——

表 1：当 ζ 为偶数时，关于起始理想和终止理想的分类讨论。分别在 $P_i^{\min}$ 中选择元素x，或在 $P_{\text{最大}i}$ 中选择元素，均可任意。

在 $\mathcal{F}^3(P_i)$ 中，以 $\varphi_i^P(I_i^f)$ 开始并以 $\varphi_i^P(I_i^l)$ 结束。连接相应的原像，便得到 $\mathcal{F}^3(P)$ 中从 I 到 P 的一条哈密顿路径。

现在假设 ζ 为奇数。我们按以下顺序处理偏序集

$$P_\zeta, P_{\zeta-2}, \ldots, P_1, P_0, P_2, \ldots, P_{\zeta-1}, P_{\zeta+1}, P_{\zeta+2}, \ldots, P_k. \tag{2}$$

对于 $i \in \{0, \ldots, k\}$，我们像之前一样定义起始理想和终止理想 I_i^f 和 I_i^l。与前一种情况相比，我们只修改 $i \in \{0, 1\}$ 的定义，因为相应的偏序集在（1）和（2）中按相反顺序遍历。我们设

$$I_1^f = D_3 \setminus \{c_1, c_2\} \text{ and } I_1^l = \begin{cases} U_1 & \text{if } \neq I_1^f \text{ or } P_1 = \emptyset \\ U_1 \setminus \{x\} & \text{else, with } x \in P_1^{\max} \end{cases},$$

以及

$$I_0^f = \begin{cases} U_1 \setminus \{c_1\} & \text{if } \neq U_0 \text{ or } P_0 = \emptyset \\ U_1 \setminus \{c_1, x\} & \text{otherwise, with } x \in P_0^{\max} \end{cases} \quad \text{and } I_0^l = U_0.$$

与偶数情形一样，可以证明，连续列出的理想至多相差三个元素，并且每次递归调用都以空集或整个偏序集开始或结束，因此我们通过归纳法得到 I 到 P 中 $\mathcal{F}^3(P)$ 的一条哈密顿路径。

情形 2： $\zeta = k$。在这种情况下，根据引理 3.3 中的对应关系，哈密顿路径以属于P_k 的理想开头和结尾。因此，我们需要将 $\mathcal{I}(P_k)$拆分，在枚举开始时遍历其中一部分，在结束时遍历另一部分。为此，我们固定 $y \in P \setminus I$ 并令$D_y = \{u \in P_k \colon \ u \preceq y\}$为 P_k中 y的下集。我们定义两个子偏序集 $P' = \{u \in P_k \colon \ u \not\succeq y\}$和 $P'' = \{u \in P_k \colon \ u \not\preceq y\}$。令 $\hat{\mathcal{I}}(P'') = \{J \cup D_y \colon \ J \in \mathcal{I}(P'')\}$。容易验证

$$\mathcal{I}(P_k) = \mathcal{I}(P') \dot{\cup} \hat{\mathcal{I}}(P'').$$

若 k 为偶数，我们按以下顺序枚举 P 的理想

$$\mathcal{I}(P'), \mathcal{I}(P_{k-2}), \mathcal{I}(P_{k-4}), \ldots, \mathcal{I}(P_0), \mathcal{I}(P_1), \mathcal{I}(P_3), \ldots, \mathcal{I}(P_{k-1}), \hat{\mathcal{I}}(P'').$$

若 k 为奇数，我们采用以下顺序

$$\mathcal{I}(P'), \mathcal{I}(P_{k-2}), \mathcal{I}(P_{k-4}), \ldots, \mathcal{I}(P_1), \mathcal{I}(P_0), \mathcal{I}(P_2), \ldots, \mathcal{I}(P_{k-1}), \mathcal{I}(P'').$$

我们使用与情形 $\zeta < k$ 中相同的 P_0、.....、P_{k-1} 起始理想和终止理想。 P' 中枚举的第一个理想是 $I'^f = I$，最后一个是

$$I'^l = \begin{cases} D_k & \text{if } \neq I \text{ or } P' = \emptyset \\ D_k \cup \{x\} & \text{else, where } x \in P'^{\min}. \end{cases}$$

P'' 中枚举的第一个理想是

$$I''^f = \begin{cases} P & \text{if } P'' = \emptyset \\ P \setminus \{x\} & \text{else, with } x \in P''^{\max}, \end{cases}$$

最后一个是 $I''^l = P$。据此，我们按照前一种情形继续进行。 □

现在，我们说明如何将前一个证明中递归顺序的选择纳入算法 1.1。为清楚起见，我们仅着重介绍上述证明所给规则的实现。

我们修改算法 枚举 1.1 中的 例程，使其接受四个输入参数：偏序集 P、输入偏序集中的元素 R 、首次访问的理想 I^f 以及 I^l 中最后访问的理想 P。递归函数会遍历 $R \cup I$ 的所有理想 I。 P

我们初始化 $D \leftarrow \bigcup_{i=2}^{\zeta} L_i$（因此 $D = D_\zeta$）、 $R' \leftarrow R \cup D$、 $P' \leftarrow \bigcup_{i=2}^{\zeta} S_i \setminus D$（因此 $P' = P_\zeta$）以及 $J^f \leftarrow I^f \setminus D$（因此 $J^f = I^f_\zeta \setminus D_\zeta$）。然后，我们通过调用 Enumerate($P'$, R', J^f, J^l)来枚举 P_ζ中的理想。

To e枚举对应于 $P_{\zeta-2}$ 的理想时，我们按如下方式更新变量：

$$
\begin{aligned}
P' &\leftarrow (P' \setminus (S_{\zeta-1} \cup S_{\zeta-2})) \cup (L_{\zeta-1} \cup L_{\zeta-2}) \ \text{// } P' = P_{\zeta-2} \\
J^f &\leftarrow (S_{\zeta-1} \cup S_{\zeta-2}) \setminus \{c_{\zeta-1}, c_{\zeta-2}\} \ \text{// } J^f = I^f_{\zeta-2} \setminus D_{\zeta-2} \\
J^l &\leftarrow \begin{cases} \emptyset & \text{if } J^f \neq \emptyset \text{ or } P' = \emptyset \\ \{x\} & \text{else, where } x \in P'^{\min}. \end{cases} \\
R' &\leftarrow R' \setminus (S_{\zeta-1} \cup S_{\zeta-2}) \ \text{// } R' = R \cup D_{\zeta-2} \\
D &\leftarrow D \setminus (S_{\zeta-1} \cup S_{\zeta-2}) \ \text{// } D = D_{\zeta-2}.
\end{aligned}
$$

接下来，我们调用 枚举$(P',\ R',\ J^f,\ J^l)$ 。引理 3.14的证明中描述的后续更改可以用类似的方式实现。

要确定 P_i中的极小或极大元素 x，我们利用针对 P_i的子节点迭代中最长链的计算，并分别令 x为这条链的极小或极大元素。与引理 3.6 中一样，我们可以证明， P 中的每个元素都只会被插入或删除常数次。因此，这个修改版算法 1.1 的单次迭代的时间复杂度为 $\mathcal{O}(n+q)$。在复杂度分析（引理 3.10）中，由于第二种情况下将偏序集 P_k拆分，我们用 $\Phi(P') + \Phi(P'')$ 替换 $\Phi(P_k)$。由于 $\Phi(P_k) \leq \Phi(P') + \Phi(P'')$，引理 3.10 的证明可以照常进行。这表明该算法具有常数摊还延迟。根据定理 2.11，我们可以将其转换为一种算法。总结如下：

定理 3.15。 按照上述递归顺序，算法 *1.1* 枚举偏序集的所有理想 P ，且具有常数延迟，使得连续访问的两个理想至多相差三个元素。初始化需要 $\mathcal{O}(n(n+q))$ 的空间和时间。

4 反链的常数摊销时间枚举

本节介绍一种递归算法（算法 1.2），该算法以常数延迟枚举输入偏序集（$P,\ \preceq$）的所有反链。它与上一节介绍的理想枚举算法密切相关，但采用了更简单的递归结构。同样，我们先介绍一个基本版本，然后说明如何将其改进为按格雷码顺序列出反链的算法。

递归结构 我们在 P 中固定一条最长链 C，并记 $C = \{c_1, \ldots, c_k\}$，其中 $c_1 \prec \cdots \prec c_k$。注意， P 的每个反链至多包含 C 中的一个元素。对于 $i = 1,\ldots,k$，我们令 P_i 表示 P 中与 c_i 不可比的所有元素组成的集合，并令 $P_r = P \setminus C$。对于 $i = 1, \ldots, k$，包含 c_i 的 P 的反链与 P_i 中的反链之间存在一个自然双射。此外，不包含 C 中任何元素的 P 的反链，恰好就是 P_r 的反链。因此，这建立了 P 的反链与 $P_1, \ldots, P_k, P_r$ 的所有反链之并之间的自然双射，我们将利用这一点进行递归。

算法描述 在初始化阶段，我们重新标记 P 的元素，以确保 $u \preceq v$ 蕴含 $u \leq v$。随后，我们按以下方式递归进行：为了构造一个反链，我们维护一个数组 A，其中存放父迭代中已加入当前待构造反链的元素。在每次迭代中，我们首先在当前考虑的偏序集 P 中找到一条最长链 C，并构造下文定义的辅助集 L_i 和 S_i。由于每个反链至多包含 C 中的一个元素，我们区分 C 中包含的是哪个元素（如果有的话）。也就是说，我们先将元素 c_1 加入 A，并对由 P 中与 c_1 不可比的元素组成的子偏序集调用递归。接着，我们以类似的方式处理元素 $c_2, \ldots, c_k$。接下来需要枚举不包含 C 中任何元素的反链。为此，我们以当前反链 A 为起点，在由 $P \setminus C$ 中的元素组成的子偏序集上调用递归。

实现细节 算法 1.2 的输入是一个偏序集 P，其上有 n 个元素。如第 3 节所述，P 由矩阵 M 给出，其中 $M_{ij} = 1$ 当 $i \prec j$ 时取值为 $M_{ij} = 0$，否则取值为 P。在预处理步骤中，我们重新标记 $i \preceq j$ 中的元素，使得 $i \leq j$。对于递归，注意到所有元素的初始顺序在每个子偏序集中都保持不变。因此，递归函数调用可以使用同一个矩阵 M 进行比较。已访问的反链存储在数组中。如第3节所述，我们设

$$s_u = \max\{i \in \{0, \ldots, k\} \colon c_i \prec u\} \text{ and } l_u = \min\{i \in \{0, \ldots, k\} \colon u \prec c_i\}.$$

对于 $i \in \{1, \ldots, k\}$，我们设 $s_{c_i} = l_{c_i} = i$。对于 $i \in \{0, \ldots, k\}$，令

$$S_i = \{u \in P \setminus C \colon s_u = i\} \text{ and } L_i = \{u \in P \setminus C \colon l_u = i\}$$

最长链（第 5 行）以及集合 L_i 和 S_i（第 6 行）按照第3节所述，使用算法 3.1 计算。

复杂度分析 现在分析算法 1.2 的时间和空间复杂度。特别是，我们将证明它实现了常数摊还延迟。我们首先分析初始化和单次迭代的时间复杂度。同样，令 Q 表示 P 中不可比对的集合，并设 $q = |Q|$。

引理 4.1。 算法*1.2* 的初始化需要 $\mathcal{O}(n^2)$ 时间。单次迭代的时间复杂度为 $\mathcal{O}(n + q)$。

证明。 初始化与算法1.1相同。最长链的计算（第 5 行）以及 s_u 和 l_u（第 6 行）中的值，都可以在 $\mathcal{O}(n + q)$ 时间内完成（参见第 3 节）。其余非递归步骤可在 $\mathcal{O}(n + q)$ 时间内完成。因此，算法 1.2 中递归函数的一次调用的时间复杂度为 $\mathcal{O}(n + q)$。 □

与算法1.1类似，可以证明以下结论：

引理 4.2. 算法*1.2*的最大递归深度为 n。递归树中从根节点到叶节点的路径上各次迭代的时间复杂度为 $\mathcal{O}(n(n + q)) \subseteq \mathcal{O}(n^3)$，使用 $\mathcal{O}(n^2)$ 内存。

现在我们开始证明，算法1.2 实现了常数摊还延迟。与第 3 节一样，我们需要对出现的集合进行更细致的分解。

分别用 n_i 和 n_r 表示 P_i 和 P_r 的大小。对于 $i = 1, \ldots, k$，令 Q_i 表示满足 u 的不可比对 $\{u, v\}$ 的集合，并令 $q_i = |Q_i|$。类似地，我们定义 Q_r 和 q_r。按照第 3 节的方式定义一个划分 $Q = Q' \dot\cup Q'' \dot\cup Q'''$：令 Q' 为 P 中包含 C 的元素的不可比对集合。令 Q'' 为 $P \setminus C$ 中在 C 上具有共同不可比元素的不可比对。最后，令 Q''' 表示其余的不可比对，即无法通过 C 中的任何元素扩展为大小为 3 的反链的不可比对。分别用 q'、q'' 和 q''' 表示 Q'、Q'' 和 Q''' 的大小。令 $Q''_i = Q'' \cap Q_i$ 和 $Q'''_i = Q''' \cap Q_i$。注意，$Q'' \cup Q''' = Q_r$ 且 $Q' \cap Q_i = \emptyset$。

令 T 为 P 中所有 3-反链的集合。如上所述，我们定义划分 $T = T' \dot\cup T'' \dot\cup T'''$，并令 t'，t''，t''' 分别表示 T'，T''，T''' 的大小。对于 $i = 1, \ldots, k$，令 $T_i \subseteq T$ 表示所有元素均属于 P_i 的那些 3-反链，并设定 $t_i = |T_i|$。类似地，我们定义 T_r 和 t_r。令 $T''_i = T'' \cap T_i$ 和 $T'''_i = T''' \cap T_i$，并定义 $t''_i = |T''_i|$ 和 $t'''_i = |T'''_i|$。类似地，我们定义 T''_r，T'''_r 和 t''_r，t'''_r。

引理 4.3。有 $q''' \le 96(\sum t''_i + \sum n_i)$。

证明。尽管递归集 $P_0, \ldots, P_k, P_r$ 的定义与第 3 节中的定义不同，但引理 3.9 的证明思路同样适用于此处。□

设 $\alpha = 196$，$\beta = 195$，$\gamma = 97$，$\delta = 96$，$\mu = 392$，并定义势函数 $\Phi(P) = \alpha + \beta n + \gamma q + \delta t$。利用该势函数验证金字塔条件（见定义 2.2），现在证明算法1.2 具有常数摊还延迟。

引理 4.4。对于每个偏序集 P满足 $n \ge 2$，有 $\sum \Phi(P_i) + \Phi(P_r) - \Phi(P) \ge n + q$。若 $n = 1$，则有 $\mu - \Phi(P) \ge 1$。特别地，算法 *1.2* 具有常数摊还延迟。

证明。我们证明，$\sum \Phi(P_i) + \Phi(P_r) - \Phi(P) - n - q \ge 0$ 当 $n \ge 2$ 时成立。我们首先分析该表达式中对应于偏序集元素的各项所作的贡献。我们有 $n = n_r + k$，因为 P中的每个元素要么包含在 C 中，要么属于 $P_r = P \setminus C$。此外，我们有 $n \le k + \sum n_i$，因为$P \setminus C$ 中的每个元素至少属于一个子偏序集 P_i。综上可得

$$\beta(n_r + \sum n_i) - (\beta + 1)n \ge (\beta - 1)\sum n_i - (\beta + 1)k.$$

接下来，我们分析对应于不可比对的项的贡献。我们有$q' = \sum n_i$，因为对于每个不可比对 $\{c, v\}$，满足 $c \in C$且 $v \in P \setminus C$，元素 v属于P_i。显然，我们有 $q'' = q''_r$。此外，由于 Q_r 中的每个不可比对都是 $P \setminus C$ 中的不可比对，反之亦然，因此我们得到 $q''' = q'''_r$。最后，我们有 $q'' \le \sum q''_i$，因为每个不可比对 u，$v \in P \setminus C$ 都与某个 c_i 属于 Q''_i。结合引理 4.3，可得

$$\gamma(q_r + \sum q_i) - (\gamma + 1)q \ge (\gamma - 1)\sum q''_i - (\gamma + 97)\sum n_i - 96\sum t''_i.$$

现在我们分析对应于 3-反链的项的贡献。我们有 $t' = \sum q_i''$，因为根据定义， $t_i' = q_i''$。显然，我们有 $t'' = t_r''$。此外，我们有 $t''' = t_r'''$，因为 Q_r 中的每个 3-反链都是 $P \setminus C$ 中的反链，反之亦然。结合这些等式，我们得到

$$\delta(t_r + \sum t_i - t) \geq \delta(\sum t_i'' - \sum q_i'').$$

结合上述不等式可得

$$\begin{aligned}
&\Phi(P_r) + \sum \Phi(P_i) - \Phi(P) - n - q \\
&\quad \geq \alpha k + (\beta - 1) \sum n_i - (\beta + 1)k + (\gamma - 1) \sum q_i'' - (\gamma + 97) \sum n_i \\
&\qquad - 96 \sum t_i'' + \delta(\sum t_i'' - \sum q_i'') \\
&\quad \geq k(\alpha - \beta - 1) + \sum n_i(\beta - \gamma - 98) + \sum t_i''(\delta - 96) + \sum q_i''(\gamma - \delta - 1) \\
&\quad = 0,
\end{aligned}$$

从而 $\Phi(P_r) + \sum \Phi(P_i) - \Phi(P) \geq n + q$。如果 $n = 1$，根据引理 4.1，算法 1.2 的一次迭代可在 $\mathcal{O}(n + q)$ 时间内执行。令 $\mu \geq \alpha + \beta + 1$。那么，如果 $n = 1$，有$\mu - \Phi(P) \geq 1$。由此以及上式，势函数 Φ 满足关于 $T^\star = \mathcal{O}(1)$ 的金字塔条件。根据定理 2.4，算法 1.2 具有常数摊还延迟。

□

注 4.5。 类似于例 *3.13*，可以证明不能使用 *Uno* 的推出条件来证明算法 *1.2* 实现常数摊还延迟。

与理想的情形类似，我们现在改进算法 1.2 ，以按格雷码顺序列出反链。对于 $k \in \mathbb{N}$，令 $\mathcal{F}^k(P)$ 为集合 P上的反链的翻转图，其中当两个反链至多相差 k个元素时，它们之间由一条边连接。容易看出，当 $\mathcal{F}^1(P)$ 是一个至少含有三个元素的链时，不存在哈密顿路径。如果 P 是一个至少含有三个元素的链，则与理想的情形不同，目前尚不清楚 $\mathcal{F}^2(P)$ 是否包含哈密顿路径。现在我们给出一个简单证明，说明$\mathcal{F}^3(P)$ 具有一条哈密顿路径，随后将其转化为一个算法。

引理 4.6。 对于每个非空偏序集 P，图 $\mathcal{F}^3$（P）都包含一条从 *1*-反链到空反链的哈密顿路径。

证明。 我们通过对 P的大小进行归纳法来证明该命题。若 $|P| = 1$，该命题显然成立，因此假设 $|P| \geq 2$。设 C是 P中的一条最长链。对于每个 $i = 1, \ldots, |C|$，我们在 $\mathcal{F}^3(P_i)$中得到一条哈密顿路径$(A_1^i, \ldots, A_{t_i}^i)$，它从空反链出发，以某个 $a_i \in P_i$对应的 1-反链 $\{a_i\}$ 结束。于是，$(\{c_i\} \cup A_1^i, \ldots, \{c_i\} \cup A_{t_i}^i) = (\{c_i\}, \ldots, \{c_i, a_i\})$是 P中所有包含 c_i的反链的一个列举。类似地，我们在$\mathcal{F}^3(P_r)$中得到一条哈密顿路径$(A_1^r, \ldots, A_{t_r}^r)$，它从 1-反链 $\{a_r\}$ 出发，以空反链结束。将这些列举连接起来，我们得到一条哈密顿路径

$$(\{c_1\}, \ldots, \{c_1, a_1\}, \{c_2\}, \ldots, \{c_{|C|}, a_{|C|}\}, \{a_r\}, \ldots, \emptyset)$$

它从 $\mathcal{F}^3(P)$中的 1-反链 $\{c_1\}$出发，到达空反链。注意，若 P_i为空，则只会经过反链 $\{c_i\}$ 。类似地，若 $P \setminus C$ 为空，则在最后一步中只会经过空反链。

□

由于引理证明中的归纳法 4.6 使用了算法 1.2的递归结构，我们可以通过选择特定的递归调用顺序来改进算法 1.2 ，使其按照此顺序访问反链。其实现以及时间复杂度得以保持的论证，与第 3.3 节中针对理想的相应推理类似。综上，我们得到以下结果：

定理 4.7. 按照上述递归顺序，算法 *1.2*以常数延迟枚举偏序集的所有理想 P ，使得连续访问的两个理想至多相差三个元素。它需要 $\mathcal{O}(n(n+q))$ 空间，并需要 k 时间进行初始化。前 $\mathcal{O}(k+n(n+q))$ 个反链的访问时间为该值。

致谢

Sofia Brenner 获得了德国学术交流中心（DAAD）的博士后奖学金资助。

参考文献

[Abd09] M. Abdo，《偏序集理想的格雷码顺序高效生成》，《信息处理快报》**109** (2009)，第13期，687–689页。[Abd10] M. Abdo，《偏序集理想的格雷码顺序高效生成》，博士论文，魁北克大学蒙特利尔分校，2010年。[Abd13] M. Abdo，《偏序集理想的格雷码顺序高效生成，第二部分》，《理论计算机科学》**502**(2013)，30–45页。[AF96] D. Avis和K. Fukuda，《用于枚举的反向搜索》，《离散应用数学》**65** (1996)，第1–3期，21–46页。[BCM+26] S. Brenner、J. Cardinal、T. McConville、A. Merino和T. M¨utze，《遍历超可解超平面排列及其格商的区域》，《2026年ACM-SIAM离散算法年度研讨会论文集》（SODA 2026），2026年，第4953–4968页。[BW91] G. Brightwell和P. Winkler，《计算线性扩张的数量是#P 完全的》，《第二十三届ACM计算理论年度研讨会论文集》（STOC 1991），1991年，第175–181页。

[CBNT21] S. Coumes、T. Bouadi、L. Nourine和A. Termier，《天际线群是理想：枚举天际线群的高效算法》，《组合算法——第32届国际研讨会》（IWOCA 2021），2021年，第223–236页。[CGM+22] A. Conte、R. Grossi、A. Marino、T. Uno和L. Versari，《极大子图枚举的邻近搜索》，《SIAM计算杂志》51 (2022)，第5期，1580–1625页。[CHMM23] J. Cardinal、H. P. Hoang、A. Merino和T. M¨utze，《在弦图和超图的无环定向中曲折前行》，《2023年ACM-SIAM离散算法研讨会论文集》（SODA 2023），2023年，第3029–3042页。[CLRS09] T. H. Cormen、C. E. Leiserson、R. L. Rivest和C. Stein，《算法导论》，第三版，麻省理工学院出版社，2009年。[CMM24] J. Cardinal、A. Merino和T. M¨utze，《通过置换语言进行组合生成。IV. 消去树》，《ACM算法汇刊》21 (2024)，第1期，1–41页。[CS23] F. Capelli和Y. Strozecki，《枚举算法的几何摊还》，第40届计算机科学理论问题国际研讨会（STACS 2023），第254卷，2023年，第18:1–18:22页。[CU19] A. Conte和T. Uno，《通过邻近搜索枚举极大子图的新多项式延迟界》，《第51届ACM SIGACT年度计算理论研讨会论文集》，STOC 2019，2019年，第1179–1190页。[CW95] E.R. Canfield和S. G. Williamson，《生成偏序集线性扩张的无环算法》，Order 12 (1995)，57–75页。[Fin25] J. Fink，《完美二分匹配的常数时间枚举》，arXiv:2509.16135 (2025)。[Flo67] R. W. Floyd，《非确定性算法》，《ACM杂志》14 (1967)，第4期，636–644页。[FM94] K. Fukuda和T. Matsui，《求出二分图中的所有完美匹配》，《应用数学快报》7 (1994)，第1期，15–18页。[GMN24] P. Gregor、T. M¨utze和Namrata，《通过置换语言进行组合生成。VI. 二叉树》，《欧洲组合学杂志》122 (2024)，104020。[Gra53] F. Gray，《脉冲编码通信》，1953年，1953年3月17日（1947年11月提交）。美国专利2,632,058。

[HHMW20] E. Hartung、H. P. Hoang、T. M¨utze 和 A. Williams，通过排列语言进行组合生成，《2020年ACM-SIAM离散算法研讨会论文集》（SODA 2020），2020年，第1214–1225页。MR 4141256[HM21] H. P. Hoang 和 T. M¨utze，通过排列语言进行组合生成。II. 格同余，《以色列数学杂志》244（2021），第1期，359–417。MR 4344032[HMNS01] M. Habib、R. Medina、L. Nourine 和 G. Steiner，分配格上的高效算法，《离散应用数学》110（2001），第2期，169–187。[HNS97] M. Habib、L. Nourine 和 G. Steiner，区间序理想的格雷码，《算法杂志》25（1997），第1期，52–66。[Joh63] S. Johnson，通过相邻换位生成排列，《数学计算》17（1963），282–285。MR 0159764（28 #2980）[KR93] Y. Koda 和 F. Ruskey，森林偏序集理想的格雷码，《算法杂志》15（1993），324–340。[Law79] E. L. Lawler，用于排序问题的动态规划算法的高效实现，BW（丛书）。数学中心基金会（1979）。[MM23a] A. Merino 和 T. M¨utze，通过排列语言进行组合生成。III. 矩形剖分，《离散与计算几何》70（2023），第1期，51–122。MR 4598046[MM23b] A. Merino 和 T. M¨utze，通过优化遍历组合0/1 多胞体，IEEE计算机科学基础第64届年会（FOCS 2023），2023年，第1282–1291页。[MM24] A. Merino 和 T. M¨utze，通过优化遍历组合0/1 多胞体，《SIAM计算杂志》53（2024），第5期，1257–1292。[MN94] R. Medina 和 L. Nourine，偏序集理想的高效生成算法，《巴黎科学院报告》S辑，数学319（1994），1115–1120。[M¨ut23] T. M¨utze，组合格雷码——最新综述，《电子组合学杂志》30（2023），3。[PB83] J. S. Provan 和 M. O. Ball，割计数的复杂度及计算图连通概率的复杂度，《SIAM计算杂志》12（1983），第4期，777–788。[Pow25] M. Powers，偏序集理想生成专题，博士论文，卡罗来纳州立大学，2025年。

[PR93] G. Pruesse 和 F. Ruskey，来自反拟阵的格雷码，Order **10** （1993），第3期，239–252。[PR94] ，快速生成线性扩张，《SIAM计算杂志》**23**（1994），第2期，373–386。[RT75] R. C. Read 和 R. E. Tarjan，列举环、路径和生成树的回溯算法界，Networks **5**（1975），第3期，237–252。[SB78] L. Schrage 和 K. R. Baker，带优先约束的排序问题的动态规划解法，《运筹学》**26**（1978），444–449。[Squ95] M. B. Squire，偏序集理想的枚举，博士论文，卡罗来纳州立大学，1995年。[Ste64] H. Steinhaus，初等数学中的一百个问题，Basic Books, Inc., Publishers，纽约，1964年。MR 0157881[Ste86] G. Steiner，生成偏序理想的一种算法，《运筹学快报》**5** （1986），第6期，317–320。[Tro62] H. F. Trotter，算法*115*：*Perm*，《ACM通讯》**5**（1962），第8期，434–435。[Uno03] T. Uno，减少枚举算法延迟和改变枚举算法的两种通用方法，技术报告，日本国立信息学研究所，东京（2003）。[Uno15] ，通过摊还实现常数时间枚举，算法与数据结构研讨会，Springer，2015年，第593–605页。[Wil14] M. Wild，输出多项式时间枚举偏序集的所有固定基数理想，以及树的所有固定基数子树，Order**31**（2014），121–135。