arXiv:2304.01744v1 [cs.DS] 4 Apr 2023

动态树宽*

Tuukka Korhonen† Konrad Majewski‡ Wojciech Nadara§ Michał Pilipczuk¶Marek Sokołowski‖

摘要

我们提出了一种数据结构，对于通过边插入和删除进行更新的动态图 G，在承诺 G 的树宽始终不超过 k的条件下，维护 G 的宽度至多为 $6k+5$ 的树分解。摊销更新时间为 $\mathcal{O}_k(2^{\sqrt{\log n_{对数对数}}})$，其中$n$ 是 G 的顶点数，而 $\mathcal{O}_k(\cdot)$记号隐藏了依赖于 k的因子。此外，我们还得到了 Courcelle 定理的动态变体：对于任何可由 CMSO_2 逻辑表达的固定性质 φ，该数据结构都能在相同的时间复杂度界内维护 G 是否满足 φ。在很大程度上，这回答了 Bodlaender 在 WG 1993]上提出的问题 [。

*Konrad Majewski、Wojciech Nadara、Michał Pilipczuk 和 Marek Sokołowski 对本稿件的工作属于一个获得欧洲研究委员会（ERC）资助的项目，资助协议编号为 948057——BOBR。Tuukka Korhonen 获得了挪威研究委员会通过 BWCA 项目（资助编号 314528）提供的支持。

†卑尔根大学信息学系，挪威（tuukka.korhonen@uib.no）

‡华沙大学信息学研究所，波兰（k.majewski@mimuw.edu.pl）§华沙大学信息学研究所，波兰（w.nadara@mimuw.edu.pl）

¶波兰华沙大学信息学研究所 (michal.pilipczuk@mimuw.edu.pl)‖波兰华沙大学信息学研究所 (marek.sokolowski@mimuw.edu.pl)

1 引言

树宽。树宽是一种图参数，用于衡量图的结构与树的相似程度。这种相似性通过树分解的概念来表达：它是一棵由袋组成的树 $\mathcal{T}$——每个袋都是顶点的子集——对于每个顶点 u，包含 u 的袋构成 $\mathcal{T}$的一棵连通子树；对于每条边 uv，则存在一个同时包含 u 和 v的袋。树分解的复杂度度量是其宽度——袋的最大规模减 1；图的树宽则是其树分解可能达到的最小宽度。

树宽和树分解（形式如上所述）最早由 Robert- son 和 Seymour 在其《图的极小割集》系列中提出，具体见 [50], ，并作为描述和分析图结构的关键组合工具。然而，这些概念很快就被发现对算法设计也极为重要。这是因为许多在一般图上难以解决的问题，例如独立集、3-着色或哈密顿性，都可以通过动态规划在有界树宽图上高效求解。典型做法是以自底向上方式处理给定的树分解，同时为分解中的每个节点 x 维护一张表，记录在 x的袋上具有不同“指纹”的部分解是否存在。指纹的数量通常是袋规模的（至少）指数函数，因此，获得宽度较小的树分解对于设计高效算法至关重要。关于这一框架的全面介绍，请参阅 Cygan 等人的教材 [24, 第 7 节。3] 。

树分解上的动态规划是参数化复杂性中的基本技术之一，在这一研究领域的各个分支中都得到广泛应用。通常，基于树分解的过程会作为更大算法中的重要子程序，用于解决那些已被简化为具有有界树宽的实例；有关这一原则的若干示例，见 [24, 第 7.7 节和 7.8]。树宽的组合性质，例如由网格图子式定理 [51], 所表达的性质，可用于分析当图的树宽较大时，图中可能存在哪些有用结构。

除参数化算法之外，树分解上的动态规划也用于近似算法，尤其是针对平面问题和几何问题的近似方案。它是基础 Baker 技术 [10], 以及无数其他近似方案的基石；这些方案遵循这样的原则：将输入图简化为有界树宽图，同时仅损失最优值的一小部分，然后在树分解上应用动态规划。

树分解上的动态规划过程可以半自动地设计。20 世纪 80 年代末，Courcelle 发现，只要一个图问题能够用 CMSO_2 ——一种带有边量化和模计数谓词的二阶单子逻辑——表达出来，那么该问题就能在树宽至多为 k 的图上，以 $\cdot n(k) \cdot n$ 的时间解决，其中 f [22] 是一个可计算函数。这里，CMSO_2 是一种强大的逻辑，它通过允许对顶点子集和边子集进行量化来扩展一阶逻辑；因此，它能够简洁地表达诸如前述 3-着色或哈密顿性等难题。有关 Courcelle 定理和 CMSO_2 的更广泛讨论，请参见 [24,第 7 节。4], [25, 第 13], 章和 [31, 第 11] 章。

在 Courcelle 定理的证明中，人们将给定的语句 φ 从 CMSO_2 转换为适当的树自动机 $\mathcal{A}_\varphi$，并在输入图的树分解上运行 $\mathcal{A}_\varphi$。因此，Courcelle 定理不仅是算法设计者的一种强大元技术，还将各种与逻辑和树自动机相关的工具带入有界树宽图这一领域。这在组合学、算法和有限模型论之间建立了根本性的联系，而有界树宽图的逻辑方面至今仍是一个充满活力的研究领域。关于这一主题的更广泛介绍，读者可参阅 Courcelle 和 Engelfriet [23] 合著的书籍。

计算树分解。 在对一个图的树分解应用任何动态规划过程之前，首先需要计算出这样的分解。因此，多年来，计算具有最优或近似最优宽度的树分解这一问题一直受到广泛关注。虽然精确计算树宽是 NP-困难的 [7], 但关于计算树分解的各种近似算法和固定参数算法，已有大量文献。下面列举一些重要的结果。

- Bodlaender 的精确固定参数算法 [13] 在时间 $2^{\mathcal{O}(k^3)} \cdot n$内，求出树宽为 k 的图的最优宽度树分解。最近，Korhonen 和 Lokshtanov [41] 将参数因子改进至 $2^{\mathcal{O}(k^2)}$，代价是将对 n 的依赖提高到 n^4。

- Robertson 和 Seymour 的 4-近似算法 [50]在时间 $\mathcal{O}(8^k \cdot k^2 \cdot n^2)$ 内，为树宽为 k 的图求出宽度至多为 $4k+3$ 的树分解。该算法展示了近似树宽最基本的方法，是算法设计教材中的重要内容 [24, 39], ，并为近似图的其他宽度参数（如秩宽或分支宽）提供了通用框架 [47, 48]。最近，Korhonen [40] 提出了一种树宽 2-近似算法，其运行时间为 $2^{\mathcal{O}(k)} \cdot n$。
- 就多项式时间近似算法而言，目前已知的最佳近似比为 $\mathcal{O}(\sqrt{\log k})$ [30]。已知，在假设小集合扩张猜想成立的情况下，树宽无法在多项式时间内近似到常数因子 [54]。

若要了解更多相关文献，请参阅上述著作的引言部分，尤其是 [41]。Pilipczuk [49] 的一篇综述文章也对计算树分解的经典方法作了通俗介绍。

除上述经典设定外，在其他算法范式下也研究了为给定图计算低宽度树分解的问题。例如，Elberfeld 等人 [28]提出了一种逐片对数空间 算法：该算法在时间 $n^{f(k)}$内计算出宽度最优的树分解，并使用空间 $f(k) \log n$，其中 f 为某个函数。Bodlaender 和 Hagerup [15] 研究了用于计算树分解的并行算法。

动态树宽。 在计算树分解这一问题上，有一种经典的算法范式一直鲜少受到关注，那就是动态数据结构。基本设定如下：给定一个动态图 G，它会随时间通过插入边和删除边进行更新。我们得到一个承诺： G 的树宽始终不超过 k，并希望维护 G 的树分解 $\mathcal{T}$，其宽度由 k 的某个函数界定。理想情况下，我们还希望与 $\mathcal{T}$ 一起维护任何合理的动态规划过程在 $\mathcal{T}$ 上的运行，从而获得 Courcelle 定理的动态变体：此外，该数据结构还应能够判断 G 是否满足任意给定的固定 CMSO_2 性质 φ。

这个关于动态树宽 的基本问题其实由来已久：Bodlaender 最早于 1993 [12] 提出。后来，Dvořák 等人 [26], 再次提出这一问题，随后 Alman 等人 [3], 、Chen 等人 [19], 和 Majewski 等人 [43] 也相继提出。尽管付出了诸多努力，这个问题迄今仍几乎无人涉足；据我们所知，在本工作之前，还没有已知的非平凡数据结构能将更新时间控制在相对于 n 的次线性范围内。下面回顾已有研究。

- 目前尚无已知方法能将上述任何用于计算树分解的静态算法推广到动态环境。

- 目前已有多种数据结构，可用于动态维护森林以及在其上运行的各种动态规划过程，参见例如 [4, 18, 45, 52]。遗憾的是，动态森林这一简单情形回避了动态树宽问题的主要难点：更新时需要重新构造树分解。因此，我们

看不出这些方法中的任何一种如何能够推广到树宽大于 1 的图。

- Bodlaender [12] 证明，对于树宽至多为 2 的图，可以维护宽度至多为 11 的树分解，且最坏情况下的更新时间为 $\mathcal{O}(\log n)$。这一结果还包含 Courcelle 定理的一个动态变体：对于树宽至多为 2 的图，任何可由 CMSO_2 表达的性质 φ 都可以维护，且最坏情况下的更新时间为$\mathcal{O}_\varphi(\log n)$。Bodlaender 的方法依赖于树宽至多为 2 的图的一个特定结构定理，但遗憾的是，该定理无法推广到更大的树宽值。在 [12], Bodlaender 还指出，对于 $k > 2$，在递减设置下（此时只允许删除边），更新时间 $\mathcal{O}_k(\log n)$ 是可以实现的。不过，这仍然避开了问题的主要难点，因为在这种设置下无需重建树分解。

- 独立于 Bodlaender，Cohen 等人 [21][1] 研究了 $k = 2$ 这一情形，其最坏情况下的更新时间为 $\mathcal{O}(\log^2 n)$；他们还研究了递增设置下的 $k = 3$ 情形，其最坏情况下的更新时间为 $\mathcal{O}(\log n)$。Frederickson [32] 研究了树宽至多为 k的图的性质的动态维护，但他考虑的更新是对树分解的直接操作；这同样避开了主要难点。

- Dvořák 等人 [26] 提出了一种数据结构，可在树深度至多为 d 的动态图中维护一棵消去森林（适用于树深度的分解），其最坏情况更新时间为$f(d)$，其中 f 是某个可计算的量。这里，树深度是一种图参数，直观而言，它衡量的是树分解的深度，而不是其宽度。图的树深度不会小于其树宽，且其上界为树宽乘以 $\log n$。后来，Chen 等人 [19] 将 Dvořák 等人的数据结构的更新时间改进为 $d^{\mathcal{O}(d)}$ </style}。除了数据结构之外，Dvořák 等人和 Chen 等人还提出了适用于树深度的 Courcelle 定理的动态变体：对于树深度至多为 的动态图，任何固定的、可用单子二阶逻辑2表达的[2] 性质 φ 都可以通过最坏情况更新时间 $\mathcal{O}_{\varphi,d}(1)$ 进行维护。

- 在 Alman 等人 [3] 关于动态反馈顶点集问题的早期工作的基础上，Majewski 等人 [43] 针对反馈顶点数这一参数证明了 Courcelle 定理的一个动态变体：反馈顶点数是指将图删除为森林所需删除集合的最小规模。这是另一个下界由树宽限定的参数。也就是说，他们证明，对于反馈顶点数至多为 ℓ 的动态图，任意固定的、可用 CMSO_2 表达的性质φ 都可以通过摊销更新时间 $\mathcal{O}_{\varphi,\ell}(\log n)$ 进行维护。
- 最近，Goranci 等人 [33] 给出了通用动态树宽问题上的首个非平凡结果：借助一种用于 扩展子层次结构 的动态算法，他们能够维护一个宽度为 $n^{o(1)}$ 近似的树分解，摊销更新时间为 $n^{o(1)}$，前提是图的最大度有界。不过需要注意的是，在（动态）参数化算法的背景下，这一结果几乎无法使用，因为树分解上的动态规划过程通常需要指数于分解宽度的时间。因此，该结果并未带来 Courcelle 定理的任何动态变体。

We 值得注意的是，动态树宽问题似乎处于这一新兴领域的基础层面
of p参数化动态数据结构。在这一研究方向中，目标是为参数化问题设计高效的数据
str结构，其中更新时间既根据相关参数衡量，也根据实例总规模衡量
par参数。示例见 [3, 12, 19, 21, 26, 27, 34, 37, 43, 46]
此类结果。

[1]遗憾的是，本稿作者未能查阅参考文献 [21]。我们对 [21] 内容的描述基于 Bodlaender 的描述 [12]。[2]单子二阶逻辑$_2$ 是 CMSO 的一个片段，其中不允许使用模计数谓词。

我们的贡献。 本文解决了动态树宽问题，实现了更新的次多项式摊销时间复杂度。具体而言，我们提出了一种数据结构，对于树宽为 k 的全动态图 G，能够维护 G 的常数因子近似树分解。对于任意固定的 k，摊销更新时间均为关于 n 的次多项式。由此，我们证明了树宽的 Courcelle 定理的动态变体：任何固定的 CMSO_2 性质 φ 的满足性都能在相同的复杂度界内得到维护。下文将以完整形式陈述我们的主要结果。

定理 1.1。 存在一种数据结构，对于初始化时固定的整数 $k \in \mathbb{N}$ 和通过边插入与删除进行更新的动态图 G，能够维护 G 的树分解，使其宽度至多为 $6k+5$，前提是 G 的树宽至多为 k。更准确地说，在任意时刻，该数据结构要么包含 G 的树分解，其宽度至多为 $6k+5$；要么包含标记 “树宽过大”，此时可保证 G 的树宽大于 k。该数据结构可在时间 $g(k) \cdot n$ 内，针对 k 和一个无边的 n 顶点图 G 完成初始化；之后，每次更新所需的摊销时间为 $2^{f(k)\cdot\sqrt{\log n}\log\log n}$，其中 $g(k) \in 2^{k^{\mathcal{O}(1)}}$ 和 $f(k) \in k^{\mathcal{O}(1)}$ 是可计算函数。

此外，初始化时还可以向该数据结构提供一个 CMSO_2 语句 φ；只要标记 “树宽过大” 不存在，它就能维护 G 中是否满足 φ 的信息。在这种情况下，初始化时间为 $g(k,\varphi)\cdot n$，摊销更新时间为 ~~$h(k,\varphi)\cdot$~~ $2^{f(k)\cdot\sqrt{\log n}\log\log n}$，其中 g、h 和 f 是可计算函数。

注意，$2^{f(k)\cdot\sqrt{\log n}\log\log n}$ 的摊销更新时间界也可以用另一种方式表示，以避免指数中出现一个依赖于 k 的因子与依赖于 n 的因子相乘。例如，如果 $f(k) \leqslant \sqrt{\log\log n}$，那么我们有 $2^{f(k)\cdot\sqrt{\log n}\log\log n} \leqslant 2^{\sqrt{\log n}\log\log n}$；否则，当 $f(k) \geq \sqrt{\log\log n}$ 时，我们有 $2^{f(k)\cdot\sqrt{\log n}\log\log n} \leqslant 2^{f(k)^2\cdot 2^{f(k)^2/2}}$；由此得到统一的界 $2^{f(k)\cdot\sqrt{\log n}\log\log n} \leqslant 2^{f(k)^2\cdot 2^{f(k)^2/2}} + 2^{\sqrt{\log n}\log\log n}$。通过将 $f(k)$ 与 n 的其他函数进行比较，还可以得到其他权衡。

此外，请注意，定理 1.1 的表述似乎比前文讨论的设置更强：即使树宽增长到高于 k 时，数据结构仍然持续有效，而不是基于树宽永远不会超过该值的假设。事实上，本文始终在后者这一较弱的设置下开展工作，因为利用 Eppstein 等人提出的延迟破坏不变量的更新技术，可以将其以一种通用方式提升到 定理 1.1 中讨论的更强设置。 [29]；另请参见 [20, 第 11]节。我们将在附录 B中讨论该技术如何应用于我们的具体问题。

在定理 1.1的数据结构中，我们仅以受限的方式修改所维护的树分解 $\mathcal{T}$：通过前缀重建更新。这类更新会将一个前缀[3] $\mathcal{S}$从 $\mathcal{T}$重建为新的前缀 $\mathcal{S}'$，并将 $\mathcal{T}-\mathcal{S}$ 中的所有树重新挂接到 $\mathcal{S}'$，而不对这些树进行修改。因此，虽然定理 1.1中我们只讨论了对单子二阶逻辑$_2$ 性质的维护，但实际上我们可以维护 $\mathcal{T}$上任意标准动态规划过程的运行。在附录 A中，我们介绍了一个用于树分解上动态规划的通用基于自动机的框架，该框架可以与我们的数据结构结合使用。用于验证单子二阶逻辑$_2$可表达性质的自动机只是该框架的一种实例。

因此，定理 1.1 回答了 Bodlaender 在 [12] 提出的问题，但更新时间还可以进一步优化。我们猜想，更新时间可以改进为 n中的多对数，甚至接近 $\mathcal{O}(\log n)$ 这个界；Bodlaender 对树宽为2的图达到了这一界。

应用。 在树分解上进行动态规划是无数参数化算法中的一项基础技术。因此，在有界树宽的全动态图中维护动态规划过程的可能性，为参数化动态数据结构的设计开辟了多条新途径。

[3] 我们所有的 树分解都是有根的，而树分解的前缀是其 ba 的一个祖先闭合集合 gs.

特别是，可以尝试将若干经典的基于树宽的方法推广到动态环境中。下面举例说明如何利用我们的数据结构，给出一种基于网格小图定理的小图包含测试经典双赢方法的动态变体。

推论 1.2. 设 H 是一个固定的平面图。存在一种数据结构，对于通过边插入和删除进行更新的 n 个顶点图 G，能够维护 H 是否为 G的图子式。在无边图上的初始化时间为 $\mathcal{O}_H(n)$，摊销更新时间为 $2^{\mathcal{O}_H(\sqrt{\log n \log\log n})}$。

证明。由于 H 是平面图，存在 $\ell \in \mathbb{N}$，使得 $\ell \times \ell$ 网格包含 H 作为图子式。因此，根据网格小图定理 [51],，存在 $k \in \mathbb{N}$，使得每个树宽大于 k 的图都包含 H 作为图子式。此外，很容易写出一个 CMSO_2 语句 φ_H，使得当且仅当图 G 包含 H 作为图子式时，该语句在图 G 中成立。现在只需针对树宽界 k 和语句 φ_H，建立定理 1.1 中的数据结构。请注意，如果该数据结构包含“树宽过大”标记，那么必然有 G 包含 H 作为图子式。 □

推论 1.2 只是一个简单的例子，但在树分解上进行动态规划是多种更复杂技术的基本构件；例如双维性、移位、无关顶点规则或元核化。因此，借助 定理 1.1，我们可以研究这些成果的动态对应形式。我们将在 第 8 节 中进一步讨论。

结构安排。 在 第 2 节 中，我们概述算法并介绍证明结果的关键思路。在 第 3 节 中，我们介绍符号和预备结果，包括“前缀重建数据结构”框架，以及将推出 引理 3.9（进而推出 定理 1.1）的主要引理陈述。在 第 4 节 中，我们证明关于称为“闭包”的对象的组合结果，随后将在 第 5 节 中利用这些结果构建名为“细化操作”的主要算法工具。接着，在 第 6 节 中，我们利用细化操作为动态树分解构建高度降低操作；在 第 7 节 中，我们综合这些结果并完成 引理 3.9 的证明。我们将在 第 8 节 中讨论结论和未来研究方向。在 附录 A 中，我们介绍用于动态维护树分解上动态规划的框架；在 附录 B 中，我们根据 定理 1.1 的证明，完成 引理 3.9。

2 概述

本节将概述我们的算法。我们首先在第 2.1 节对整个算法进行高层次描述，然后在第 2.2 节和2.3 节概述最重要技术要点的证明。

2.1 高层次描述

令 n 为顶点数， k 为给定参数，它是所考虑动态图 G 的树宽上界。我们的目标是维护一个高度为 $2^{\mathcal{O}_k(\sqrt{\log n \log\log n})}$、宽度至多为 $6k+5$ 的有根树分解 $\mathcal{T}$，同时在 $\mathcal{T}$ 上维护任意动态规划方案，或者更正式地说，维护一个求值时间为 $\mathcal{O}_k(1)$ 的树分解自动机。我们还要求 $\mathcal{T}$ 为二叉的，即每个节点至多有两个子节点。

维护这样的树分解是合理的，因为 Bodlaender 和 Hagerup 的一个著名引理 [15] 指出：对于每个树宽为 k 的图，都存在一个高度为 $\mathcal{O}(\log n)$、宽度至多为 $3k+2$ 的二叉树分解。于是，假设 $\mathcal{T}$ 是一个高度为 h、宽度为 $\mathcal{O}(k)$ 的二叉树分解，则可以按如下方式在时间 $\mathcal{O}_k(h)$ 内实现添加边或删除边操作。假设我们将边 uv 存在于其袋同时包含 u 和 v 的最高节点中。删除边 uv 时，只需找到 $\mathcal{T}$ 中同时包含 u 和 v 的最高节点，更新该节点中存储的这条边是否存在的信息，然后更新从该节点到根的路径上各节点的动态规划表，耗时 $\mathcal{O}_k(h)$。在顶点 u 和 v 之间执行边添加操作时，令 P_u（分别为 P_v）表示 $\mathcal{T}$ 中从包含 u（分别为 v）的最高节点到根的路径；将 u 和 v 添加到 $P_u \cup P_v$ 上的所有袋中，将边 uv 存在的信息添加到根节点，并更新 $P_u \cup P_v$ 上的动态规划表，总耗时同样为 $\mathcal{O}_k(h)$。需要强调的是，只有同时包含 u 和 v 的最高袋才会“知晓”边 uv 的存在，而不是采用更直观的做法，让所有同时包含 u 和 v 的袋都“知晓” uv。这对于确保边添加或删除后只需重新计算 $\mathcal{O}(h)$ 个节点的动态规划表至关重要。

现在，唯一的问题是，边添加操作可能会使 $\mathcal{T}$ 的宽度增加到超过 $6k+5$。通过维护 $\mathcal{T}$ 上的 Bodlaender-Klok 的动态规划 [16]，我们可以检测图的树宽是否实际上增加到超过 k，并在这种情况下终止算法[4]。更有意思的情况是，图的树宽仍然至多为 k；此时我们必须修改 $\mathcal{T}$，使其宽度减小，同时仍保持较小的高度。本文的主要技术贡献在于证明，对树分解进行此类修改确实可以高效实现。

下面引入一些记号。我们用一对 $\mathcal{T} = (T, \text{袋})$ 表示一个有根树分解，其中 T 是一棵有根树，而 $\mathsf{bag} : V(T) \to 2^{V(G)}$ 是一个函数，用于指定每个节点 t 的袋 $\mathsf{bag}(t)$。对于节点集合 $W \subseteq V(T)$，我们用 $\mathsf{bags}(W)$ 表示 $= \bigcup_{t \in W} \mathsf{bag}(t)$。有根树 T 的前缀 是包含根且诱导出连通子树的节点集合 $T_{\text{pref}} \subseteq V(T)$；有根树分解 $\mathcal{T} = (T, \text{袋})$ 的前缀则是 T 的前缀。对于有根树 T，我们用 $\mathsf{height}(T)$ 表示根到叶路径上的最大节点数；对于节点 $t \in V(T)$，$\mathsf{height}(t)$ 表示以该节点为根的子树的高度 t。

回想一下，在边添加操作中，我们增大了由两条路径的并集 $P_u \cup P_v$ 构成的子树中各袋的规模，其中每条路径都连接一个节点与根。特别地，所有袋过大的节点都包含在规模至多为 $2h$ 的前缀 $P_u \cup P_v$ 中，其中 h 是我们的树分解的高度。现在，一个自然的改进宽度的想法是，用一个高度为 $\mathcal{O}(\log n)$、宽度为 $3k+2$ 的树分解替换前缀 $P_u \cup P_v$，该树分解由 Bodlaender-Hagerup 引理给出，分解对象是 $G[\text{袋}(P_u \cup P_v)]$。虽然这个想法过于天真，但我们将展示，令人惊讶的是，我们可以实现一种精神上与之相似的做法。

我们为在动态环境中维护树分解而开发的主要工具是细化操作。该操作的定义和性质较为技术性，将在第 2.2 节中介绍；这里先非正式地说明该操作的作用。细化操作以我们正在维护的树分解的一个前缀 T_{pref} $\mathcal{T}$ 作为输入。粗略地说，它将 T_{pref} 替换为一个宽度至多为 $6k+5$、高度至多为 $\mathcal{O}(\log n)$ 的树分解。该操作还会修改 $\mathcal{T}$ 的其他部分，但修改后这些部分在袋规模方面只会变得更好。特别是，如果我们在 $T_{\text{pref}} = P_u \cup P_v$

[4]通过延迟更新这一标准技术，实际上我们无需终止算法；不过在此概述中，为简单起见，我们假设当宽度超过 k 时，可以直接终止算法。

在一次边添加操作之后，如果 $P_u \cup P_v$ 中节点的宽度超过 $6k+5$，该操作会将 $\mathcal{T}$ 的宽度恢复到至多 $6k+5$。细化操作的摊销时间复杂度为 $\mathcal{O}_k(|T_{\mathrm{pref}}|)$，并且最多会使 $\mathcal{T}$ 的高度增加 $\mathcal{O}(\log n)$。

通过细化操作，我们可以确保所维护的树分解 $\mathcal{T}$的宽度不超过 $6k+5$。然而，每次应用细化操作都会使 $\mathcal{T}$的高度增加 $\log n$，因此我们还需要一种降低高度的工具。我们结合精心选择的势函数，以及在高度过大时利用细化操作“免费”降低势函数的策略，开发了这样一种工具。具体而言，我们使用的势函数为

$$\Phi(\mathcal{T}) = \sum_{t \in V(T)} (\gamma \cdot k)^{|\mathsf{bag}(t)|} \cdot \mathsf{height}(t),$$

其中 $\gamma < 1000$ 是 第 5.1 节中定义的一个固定常数。这个函数具有以下性质：在边添加操作中不会增长过多（增长至多为 $\mathcal{O}_k(\mathsf{height}(T)^2)$）；它与细化操作的摊还分析细节相得益彰（因子 $(\gamma \cdot k)^{|\mathsf{bag}(t)|}$ 源于此处）；此外，由于包含因子 $\mathsf{height}(t)$，它自然会在高度较小的树上取较小的值。在第 2.3 节 中，我们将概述一种策略：当 $\mathcal{T}$的高度超过$2^{\mathcal{O}_k(\sqrt{\log n}\log\log n)}$时，该策略会选取一个前缀 T_{pref}，使得对 T_{pref}应用细化操作后， $\Phi(\mathcal{T})$ 的值会降低；此外，细化操作的运行时间可以由这一下降量界定。具体而言，只要高度大于 $2^{\mathcal{O}_k(\sqrt{\log n}\log\log n)}$，我们就可以从摊还运行时间的角度“免费”地应用这种细化操作，同时降低势函数的值。由于势函数不可能永远持续下降，反复应用这种操作最终会使高度降至至多 $2^{\mathcal{O}_k(\sqrt{\log n}\log\log n)}$。

2.2 细化操作

本小节概述细化操作。首先需要指出，我们的细化操作建立在 Korhonen 和 Lokshtanov 最近提出的树分解改进操作之上，该操作用于改进求解树宽的静态固定参数算法 [41], ；而这一操作又建立在 Korhonen 的树宽 2-近似算法 [40] 之上，尤其借鉴了 Thomas [53] 以及 Bellenbaum 和 Diestel [11]在计算树宽时使用的技术。我们的细化操作将 [41] 的改进操作加以推广，允许细化连续子树，而不只是单个袋；这对于控制树分解的高度至关重要。我们还将该操作从适用于精确计算树宽的形式改造为更具近似性的版本（ $6k+5$ 宽度由此与 Bodlaender-Hagerup 引理 [15]相结合），以获得高效运行时间所需的结构性质。为此，我们使用 Bojańczyk 和 Pilipczuk 的去交替引理 [17] 。本节其余部分不要求读者了解这些先前的结果。

回顾一下细化操作的目标：给定我们正在维护的树分解 $\mathcal{T}$ =(T，袋) 的前缀 T_{pref}，我们希望以某种方式在这个前缀上重新计算树分解。注意，我们不能简单地选取一个诱导子图，例如 $G[\mathsf{bags}(T_{\mathrm{pref}})]$，并期望用它的任意树分解替换 T_{pref}，因为树分解还需要考虑袋之外的顶点所提供的连通性。为此，合适的概念是顶点集的 躯干图 。设 G 是一个图， $X \subseteq V(G)$ 是一个顶点集。图 $\mathrm{torso}_G(X)$ 的顶点集为 X；若 G 中存在一条 u-v 路径，且其内部顶点均在 X 之外，则 X 中有一条边 uv。换言之，$\mathrm{torso}_G(X)$ 是由 $G[X]$ 得到的超图：对于 $G - X$ 的每个连通分量 C，将其邻域 $N(C)$ 补成一个团。

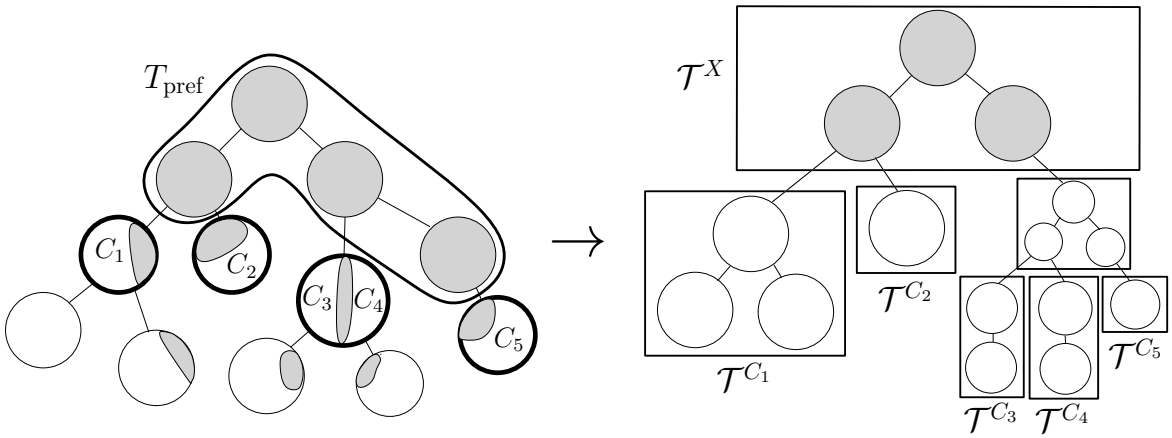


图 1：细化操作。左图展示了树分解 $\mathcal{T}$，其中前缀 T_{pref} 被圈出， $X \supseteq$ 中袋内的顶点以灰色表示。 T_{pref}的附属子树用粗体圈出，而 $G - X$ 的连通分量标记为 C_1、.....、 C_5。右图展示了通过使用 X对 $\mathcal{T}$ 进行细化后构造的树分解，具体而言，是取躯干图$_G(X)$的树分解 $\mathcal{T}^X$，并将 $G - X$ 的连通分量 C_i 各自对应的树分解 $\mathcal{T}^{C_i}$ 粘接到其上。由 $\mathcal{T}^{C_3}$、 $\mathcal{T}^{C_4}$、 $\mathcal{T}^{C_5}$ 上方三个节点组成的子树，是为在重新接上 $\mathcal{T}^{C_3}$、 $\mathcal{T}^{C_4}$、 $\mathcal{T}^{C_5}$后保持树为二叉树而构造的。

接下来我们概述细化操作（示意图见图 1）。给定 T_{pref}，我们找到一个顶点集 $X \supseteq \text{bags}(T_{\text{pref}})$，使得 $\text{torso}_G(X)$ 的树宽至多为 $2k+1$（出于稍后将解释的技术原因，这里用的是 $2k+1$ 而非 k ）。随后，我们计算 $\text{torso}_G(X)$ 的一个最优宽度树分解 $\mathcal{T}^X$，并使用 Bodlaender-Hagerup 引理 [15] 将其高度设为 $\mathcal{O}(\log n)$，从而得到宽度至多为 $6k+5$ 的 $\mathcal{T}^X$。我们将 $\mathcal{T}^X$ 的一个任意节点选作根，它将构成新的细化树分解的一个前缀。剩下的工作是为 $G - X$ 的每个连通分量 C 构造树分解 $\mathcal{T}^C$，并将它们连接到 $\mathcal{T}^X$ 中。

我们称 $\mathcal{T}$ 中的节点 $a \in V(T)$ 为 T_{pref} 的附属节点，当且仅当 a 不属于 T_{pref}，但 a 的父节点属于 T_{pref}。对于 T_{pref} 的每个附属节点 a，用 $(\mathcal{T}_a, \text{袋}_a)$ 表示将 $\mathcal{T}$ 限制在以 a 为根的子树上所得的分解，其中 T_a。现在注意到，由于 T_{pref} 的袋 $X \supseteq$，对于 $G - X$ 的每个连通分量 C，都存在唯一一个 T_{pref} 的附属节点 a，使得 C 包含在 $\mathcal{T}_a$ 的各个袋中。此外，将 $(T_a, \text{袋}_a|_{N[C]})$ 限制在 C 的闭邻域 $N[C]$ 上，得到的是诱导子图 $G[N[C]]$ 去掉 $N(C)$ 内部边后的树分解。然后注意到，由于 $\mathcal{T}^X$ 是躯干图 $_G(X)$ 的树分解，它必有一个包含 $N(C)$ 的袋。现在，我们的目标是将 $(T_a, \text{袋}_a|_{N[C]})$ 接到这个袋中。为了在满足树分解的连通性条件的同时实现这一点，我们需要在 $(T_a, \text{袋}_a|_{N[C]})$ 的根中包含集合 $N(C)$。我们用 $(\mathcal{T}^C, \text{袋}^C)$ 表示从 $(T_a, \text{袋}_a|_{N[C]})$ 得到的树分解：通过“强制”将 $N(C)$ 放入根节点 a 的袋 $^C(a)$ 中，具体做法是将 $N(C)$ 加入袋 $^C(a)$，然后为修复连通性条件，将每个顶点 $v \in N(C)$ 加入从根到包含 v 的其他袋所在子树之间的唯一路径上的所有袋中。于是， $\mathcal{T}^C$ 是 $G[N[C]]$ 的一个树分解，其根袋包含 $N(C)$，因此可以将它接到 $\mathcal{T}^X$ 中包含 $N(C)$ 的袋上。这些连接可能使所得树分解的度数超过 2，因此最后需要将这些高阶节点展开为二叉树。至此，我们对细化操作的非正式描述就结束了。实际定义稍微复杂一些，

因为为了获得高效的运行时间，有必要：(1) 在某些情况下，将 $\mathcal{T}_a$ 中多个不同的连通分量 C 视为一个连通分量；(2) 剪除 $\mathcal{T}^C$ 中一些不必要的袋。

根据上文概述的细化操作，可以清楚地看出，得到的树分解确实是 G 的树分解。也很容易看出，细化后的树分解的高度至多为 $\mathsf{height}(\mathcal{T}) + \mathcal{O}(\log n)$：这是因为 $\mathcal{T}^X$ 的高度至多为 $\mathcal{O}(\log n)$，而每个附加分解 $\mathcal{T}^C$ 的高度至多为 $\mathsf{height}(\mathcal{T})$。回想一下，我们的目标是：如果 $\mathcal{T}$ 中宽度大于 $6k+5$ 的所有袋都包含在 T_{pref} 中，那么细化后的树分解的宽度至多为 $6k+5$。显然，$\mathcal{T}^X$ 中各袋的宽度至多为 $6k+5$。然而，由于将顶点额外插入 $\mathcal{T}^C$ 中的袋 $N(C)$，这些袋的宽度为何至多为 $6k+5$ 并不明显。事实上，若没有 X 的某些附加性质，我们无法保证这一点；下面将概述这些性质。

我们称顶点集 $X \subseteq V(G)$ 是 T_{pref} 的一个 k-闭包，如果 $X \supseteq$ 袋(T_{pref})，且躯干图$_G(X)$ 的树宽至多为 $2k+1$。特别地，细化操作中的集合 X 是 T_{pref} 的一个 k-闭包。若对 $G-X$ 的每个连通分量 C，集合$N(C)$ 都以如下意义与袋(T_{pref})相连：不存在规模为 $< |N(C)|$、能将$N(C)$与袋(T_{pref})分隔开的分隔集，则称 k-闭包 X 与 T_{pref} 相连。控制 $\mathcal{T}^C$ 宽度的关键性质是：如果 X 与T_{pref}相连，那么 $\mathcal{T}^C$ 的每个袋的宽度至多为 $\mathcal{T}$ 中对应袋的宽度。特别地，设 $\mathcal{T}_a$ 是 $\mathcal{T}$ 中挂在 T_{pref} 的一个附属节点 a 上的子树，C 是包含于 $\mathcal{T}_a$ 中的 $G-X$ 的一个连通分量。回顾一下，我们通过取 $T^C = T_a$，然后对每个 $t \in V(T^C)$ 进行如下设置，来构造树分解 $\mathcal{T}^C =$(袋^C, 袋^C)

$$\mathsf{bag}^C(t) = (\mathsf{bag}_a(t) \cap N[C]) \cup \{v \in N(C) \mid \text{the highest bag containing } v \text{ is below } t\}.$$

因此，我们可以如下界定袋$^C(t)$的规模。

引理 2.1. 如果 X链接到 T_{pref}，那么 $|$袋$^C(t)| \leqslant |$袋$_a(t)|$。

证明概要。设 $N_t = \{v \in N(C) \mid$是包含 v的最高袋，位于 $t\}$之下，并注意到只需证明 $|N_t| \leqslant |\mathsf{bag}_a(t) \setminus N[C]|$。根据链接性和门格尔定理，存在 $N(C)$ 条从 $N(C)$ 到袋(T_{pref})的顶点不相交路径；由于袋(T_{pref})与 C不相交，这些路径的所有内部顶点都在 $V(G) \setminus N[C]$中。此外，这些路径中有 $|N_t|$ 条从 N_t 通向袋(T_{pref})；由于袋$_a(t)$将 N_t与袋(T_{pref})分隔开，且与 N_t不相交，因此每条这样的路径都包含一个位于袋$_a(t)$中的内部顶点，从而推出 $|\mathsf{bag}_a(t) \setminus N[C]| \geqslant |N_t|$。 □

引理 2.1 表明，如果 T_{pref} 包含所有宽度大于 $6k+5$ 的袋，并且 X 链接到T_{pref}中，那么所得的树分解的宽度至多为 $6k+5$。我们注意到，T_{pref}的k-闭包链接到 T_{pref} 中并非显而易见；不过，在深入讨论之前，我们先立即推广链接性的概念，以获得引理 2.1 的一种更强形式，这将有助于分析势函数。顶点 $v \in V(G)$ 在 $\mathcal{T} =$(T，袋) 中的深度 $d(v)$，是 T 中包含 v 的最高节点的深度（即该节点到根的距离）。对于顶点集 S，我们将 $d(S)$ 记作 $= \sum_{v \in S} d(v)$。于是，如果一个 k-闭包 X 链接到T_{pref}中，并且对于每个邻域 $N(C)$，都不存在满足 $|S| = |N(C)|$且 $d(S) < d(N(C))$ 的分隔集 S 将 $N(C)$ 与袋 (T_{pref}) 分隔开，那么我们称该闭包是 d- 链接到 T_{pref} 中的。

回顾我们的势函数 $\Phi(\mathcal{T}) = \sum_{t \in V(T)} (\gamma \cdot k)^{|\mathsf{bag}(t)|} \cdot$高度$(t)$。利用 d-链接性，我们能够证明细化操作的实际定义满足以下性质。

引理 2.2 (非正式表述)。设 X 是 T_{pref} 的 d-连通的 k-闭包，a 是 T_{pref} 的一个附属节点，而 $C_1, \ldots, C_\ell$ 是包含于 $\mathcal{T}_a$ 中的 $G-X$ 的连通分量。则有 $\sum_{i=1}^{\ell} \Phi(\mathcal{T}^{C_i}) \leqslant \Phi(\mathcal{T}_a)$；此外，对于所有 i，树分解 $\mathcal{T}^{C_i}$ 及其更新后的动态规划表都可以在 $\mathcal{O}_k(\Phi(\mathcal{T}_a) - \sum_{i=1}^{\ell} \Phi(\mathcal{T}^{C_i}))$ 时间内构造出来。

需要指出的是，引理 2.2 对于不含高度(t) 因子的势函数也成立；势函数中包含这一因子，只是为了用于下文第 2.3 节将要概述的高度缩减方案。此外，在实际的细化操作中， C_i 在引理 2.2 中实际上可以是多个具有相同邻域 $N(C_i)$的不同连通分量的并集；对于其中每个这样的“连通分量”，我们还可以从势函数中额外计入一点势能。这部分额外势能将用于为高阶附着点构造二叉树。

忽略这些繁多的技术细节后，引理 2.2 的核心结论是，构造分解 $\mathcal{T}^C$ 就势函数而言是“免费的”。唯一可能耗费大量时间或大幅增加势函数的环节，是寻找集合 X 并构造树分解 $\mathcal{T}^X$。为了给这一过程设定界，我们给出一个引理，断言可以假设 X 的规模至多为 $\mathcal{O}_k(|T_{\text{pref}}|)$，并且还满足一种更强的结构性质，这一性质在高度缩减方案中将发挥作用。对于 a 中的一个节点 $\mathcal{T}$，用 $\texttt{cmp}(a) \subseteq V(G)$ 表示出现在以 a为根的子树的各个袋中、但不出现在 a的父节点的袋中的顶点。我们利用 Bojańczyk 和 Pilipczuk 的去交替引理证明以下结论 [17]。注意，$2k+1$ 在 k-闭包定义中的界正是由此证明得出的。

引理 2.3。 设 G 是一个树宽至多为 k 的图，且 $\mathcal{T}$是 G 的一个宽度为 $\mathcal{O}(k)$ 的树分解。对于 $\mathcal{T}$的任意前缀 T_{pref}，存在 T_{pref} 的一个 k-闭包 X，使得对于 T_{pref} 的每个附属节点 a，都有 $|X \cap \texttt{cmp}(a)| \leqslant \mathcal{O}(k^4)$。

特别地，由于 $\mathcal{T}$是一棵二叉树， T_{pref}至多有 $|T_{\text{pref}}|+1$ 个附属节点。因此，根据引理 2.3， T_{pref}存在一个 k-闭包，其中顶点数至多为 $|T_{\text{pref}}| \cdot (k+1) + \mathcal{O}(|T_{\text{pref}}| \cdot k^4) = \mathcal{O}(|T_{\text{pref}}| \cdot k^4)$。利用这样的 k-闭包，我们可以将 $\mathcal{T}^X$ 的规模界定为 $\mathcal{O}_k(|T_{\text{pref}}|)$。由于在所得分解中， $\mathcal{T}^X$ 的每个节点的势函数至多为 $\mathcal{O}_k(\text{height}(\mathcal{T}) + \log n)$，因此细化操作至多使势函数增加 $\mathcal{O}_k(|T_{\text{pref}}| \cdot (\text{height}(\mathcal{T}) + \log n))$。特别地，在边插入后直接执行的细化操作中，我们有 $|T_{\text{pref}}| \leqslant 2|\text{height}(\mathcal{T})|$，因此势函数至多增加 $\mathcal{O}_k(\text{height}(\mathcal{T})^2)$（注意，$\text{height}(\mathcal{T}) \geqslant \log n$）。

现在我们来讨论两个搁置已久的问题：如何保证k-闭包 X 是 d-链接的，以及如何实际找到这样的 X。我们称一个闭包为 c-小，如果它满足引理 2.3 在某个特定界 $c \in \mathcal{O}(k^4)$ 下的条件，而该界可从引理 2.3的证明中得到。下面的引理与 Korhonen 和 Lokshtanov [41, 节 5], 中的证明类似，它给出了一个保证 d链接性的简单条件。

引理 2.4。 设 T_{pref} 是树分解的一个前缀。若 X 是 T_{pref} 的一个 c-小 k-闭包，并且在 T_{pref} 的所有 c-小 k-闭包中，首先使 $|X|$最小，其次使 $d(X)$最小，则X 是 d-链接到 T_{pref}中的。

利用引理 2.4，我们可以用动态规划来寻找 c-小且 d-链接的 k-闭包。具体来说，我们将 Bodlaender 和 Kloks [16] 用于计算树宽的动态规划，改造为计算满足引理中优化条件的 c-小 k-闭包。这种改造采用了相当标准的技术，但需要指出一个复杂之处：即使对树分解做很小的改动，也可能改变所有顶点的深度，因此我们不能只在动态规划表中存储值 $d(X)$。相反，我们必须利用函数 d 的定义，以及我们首先要最小化 $|X|$这一事实。在整个算法过程中维护这些动态规划表后，对于给定的 T_{pref}，我们便能在 $\mathcal{O}_k(|T_{\text{pref}}|)$时间内找到一个 $\mathcal{O}(k^4)$-小的 k-闭包 X，它是 d-链接到 T_{pref} 中的，同时还能找到图的躯干图$_G(X)$。

2.3 高度缩减

在此　本小节概述高度缩减方案，尤其是以下引理　　a.

引理 *2.5*（高度降低）。设 $\mathcal{T}$ 是我们正在维护的树分解。存在一个函数 $f(n,k) \in 2^{\mathcal{O}_k(\sqrt{\log n \log\log n})}$，使得如果 $\mathit{height}(\mathcal{T}) > f(n,k)$，那么存在 $\mathcal{T}$ 的一个前缀 T_{pref}，对 T_{pref} 执行细化操作后，得到一个树分解 $\mathcal{T}'$，满足 $\Phi(\mathcal{T}') < \Phi(\mathcal{T})$，且运行时间为 $\mathcal{O}_k(\Phi(\mathcal{T}) - \Phi(\mathcal{T}'))$。

为了概述引理 2.5 的证明，我们来构建一个模型，用于衡量带有前缀 T_{pref} 的细化操作中势函数的变化。首先回顾一下，引理 2.2 已经处理了 $\mathcal{T}^C$ 中各连通分量 C 的树分解的势函数，而我们唯一需要担心势函数增大的地方是 $\mathcal{T}^X$ 的节点。在上一节中，我们将这一势函数增量界定为 $\mathcal{O}_k(|T_{\mathrm{pref}}|\cdot(\mathsf{height}(\mathcal{T})+\log n))$；其中，因子 $\mathsf{height}(\mathcal{T})+\log n$ 尤其源于这样一个事实：附加树分解 $\mathcal{T}^C$ 后，$\mathcal{T}^X$ 中节点的高度可能达到 $\mathsf{height}(\mathcal{T})+\log n$。这个上界足以处理边添加后执行的细化操作，但要证明引理 2.5，我们需要更细致的分析。

考虑以下模型：我们从高度为 $\mathcal{O}(\log n)$ 的 $\mathcal{T}^X$ 的树分解开始，然后将各连通分量 $\mathcal{T}^C$ 的树分解逐一附加到它上面。每次附加树分解 $\mathcal{T}^C$ 时，$\mathcal{T}^X$ 中至多 $\mathcal{O}(\log n)$ 个节点的高度会增加（因为 $\mathsf{height}(\mathcal{T}^X) \leqslant \mathcal{O}(\log n)$），且每个节点的高度最多增加 $\mathsf{height}(\mathcal{T}^C)$，而这至多为 $\mathsf{height}(a)$；其中，a 是 T_{pref} 的附属节点，其子树包含 C。虽然 $\mathcal{T}_a$ 中可能包含许多连通分量 C，但请注意，引理 2.3 表明，这些连通分量实际上至多具有 $k^{\mathcal{O}(k)}$ 种不同的邻域 $N(C)$，因此在 $\mathcal{T}^X$ 中至多有 $k^{\mathcal{O}(k)}$ 个不同的附着点。尤其是，在处理用于展平高度较大的附着点的二叉树所涉及的技术细节后，我们可以假设，T_{pref} 的一个附属节点 a 至多负责增加 $\mathcal{T}^X$ 中 $\mathcal{O}_k(\log n)$ 个节点的高度。此外，它对这些节点的高度增量各至多为 $\mathsf{height}(a)$，因此它总共负责使势函数增加 $\mathcal{O}_k(\mathsf{height}(a)\cdot\log n)$。

记 $\Phi(T_{\mathrm{pref}}) = \sum_{t\in T_{\mathrm{pref}}} (\gamma\cdot k)^{|\mathsf{bag}(t)|}\cdot$高度$(t)$，即 T_{pref} 中节点上的势函数值。结合上述讨论与 引理 2.2，可得如下引理。

引理 2.6。设 T_{pref} 是树分解 $\mathcal{T}$ 的一个前缀，$A \subseteq V(T)$ 是 T_{pref} 的附属子树，而 $\mathcal{T}'$ 是用 T_{pref} 对 $\mathcal{T}$ 进行细化后得到的树分解。则有

$$\Phi(\mathcal{T}') \leqslant \Phi(\mathcal{T}) - \Phi(T_{\mathrm{pref}}) + \mathcal{O}_k(|T_{\mathrm{pref}}|\cdot\log n) + \sum_{a\in A}\mathcal{O}_k(\mathsf{height}(a)\cdot\log n).$$

现在，为了证明引理 2.5，只需证明：如果 $\mathcal{T}$ 的高度过大，那么存在一个前缀 T_{pref}，其附属子树为 A，使得 $\Phi(T_{\mathrm{pref}}) > c_k \log n\left(|T_{\mathrm{pref}}| + \sum_{a\in A}\mathsf{height}(a)\right)$，其中 c_k 是一个足够大的数，且取决于 k。（这里，所需的 c_k 源自附着点的数量和势函数，因此某个 $c_k = k^{\mathcal{O}(k)}$ 就足够了。）在我们的证明中，$\Phi(T_{\mathrm{pref}})$ 的值实际上会比 $c_k\log n\left(|T_{\mathrm{pref}}| + \sum_{a\in A}\mathsf{height}(a)\right)$ 大任意给定的常数倍，这也保证了对此类前缀 T_{pref} 执行细化操作的运行时间为 $\mathcal{O}_k(\Phi(\mathcal{T}) - \Phi(\mathcal{T}'))$。

我们首先概述在 $\mathsf{height}(T) > n^\varepsilon$，且对于某个 $\varepsilon > 0$，k 相较于 n 较小时，如何选择 T_{pref}；见图 2。为简单起见，假设 $|V(T)| = n$。自然的策略是先将 T_{pref} 设为从根到 T 中最深叶节点的路径。于是我们有 $\Phi(T_{\mathrm{pref}}) \geqslant \Omega(n^{2\varepsilon})$。然而，$T_{\mathrm{pref}}$ 可能有 $n^\varepsilon/2$ 个高度为 $n^\varepsilon/2$ 的附属节点，因此 $\sum_{a\in A}\mathsf{height}(a)\cdot\log n \geqslant n^{2\varepsilon}\cdot\log n/4$ 是可能的。关键观察是，在这种情况下，许多附属节点的子树 $a\in A$ 必定比 T 更加不平衡：它们的高度至少为 $n^\varepsilon/2$，而节点数至多为 $4\cdot n^{1-\varepsilon}$（这可直接由计数论证得出）。具体而言，我们称以附属节点 a 为根的子树为大型子树，当且仅当其包含超过 $c_k\cdot n^{1-\varepsilon}\cdot\log n$ 个节点；若 $\mathsf{height}(a) \leqslant n^\varepsilon/(c_k\cdot\log n)$，则称其为浅子树。现在，

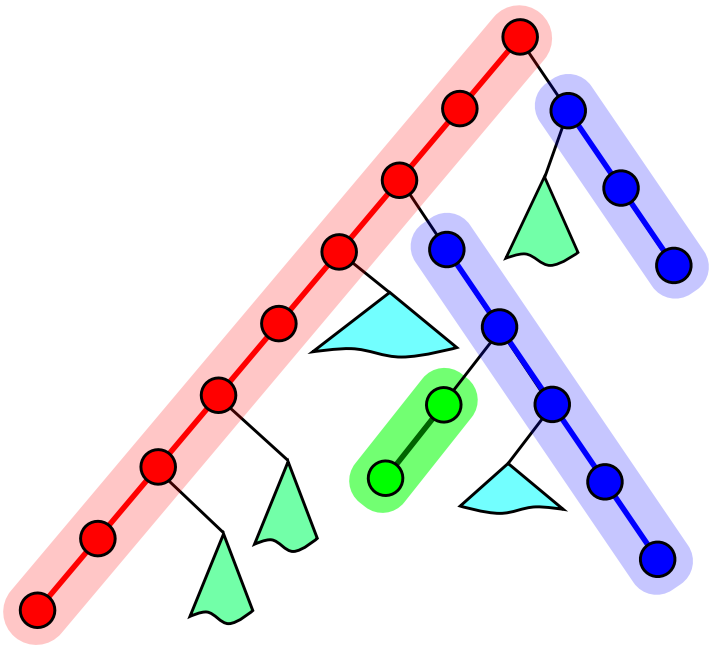


图 2：高度缩减中的 T_{pref} 构造。构造过程中提取的连续路径以红色、蓝色和绿色表示。它们的并集构成 T_{pref}。大型树以海绿色表示，浅树以青色表示。

大型子树至多有 $n/(c_k \cdot n^{1-\varepsilon} \cdot \log n) = n^{\varepsilon}/(c_k \cdot \log n)$ 个，因此它们对总和 $\sum_{a \in A} \mathcal{O}_k(\text{height}(a) \cdot \log n)$ 的贡献至多为 $\mathcal{O}_k(n^{2\varepsilon}/c_k)$。类似地，浅子树对该总和的贡献也至多为 $\mathcal{O}_k(n^{2\varepsilon}/c_k)$。因此，只要将常数 c_k 取得足够大，来自大型子树、浅子树（或两者兼具）的贡献就只占 $\Phi(T_{\text{pref}})$ 的极小一部分。

接下来，有些附属节点的子树既不大也不浅；这些子树既小又深，因此看起来比 $\mathcal{T}$ 更加不平衡。我们对这些树递归地应用相同的策略。对于每个其子树既小又深的附属节点 a，我们将从 a 到其最深后代的路径 P_a 插入到 T_{pref} 中。由于该子树很深，因此 $\Phi(T_{\text{pref}})$ 会增加 $\Phi(P_a) = \Omega(n^{2\varepsilon}/(c_k \cdot \log n)^2)$。现在，在分析 P_a 的附属节点时，我们再次应用这一策略：通过从 $\Phi(P_a)$ 收取费用来处理大型或浅的子树，然后递归处理既小又深的子树。这一次，对“大型”的恰当定义是至少包含 $n^{1-2\varepsilon}(\log n \cdot c_k)^3$ 个节点，而对“浅”的恰当定义是高度至多为 $n^{\varepsilon}/(c_k \cdot \log n)^2$。更一般地，在这种递归的第 i 层，我们可以将包含超过 $n^{1-i\varepsilon}(\log n \cdot c_k)^{i \cdot (i+1)/2}$ 个节点的子树称为大型子树，并将高度至多为 $n^{\varepsilon}/(c_k \cdot \log n)^i$ 的子树称为浅子树。当 ε 为常数时，这种递归最多只能持续常数层，因为子树不可能同时既小又深：那将要求其高度大于节点数。因此，最终我们能够找到一个满足引理 2.5 要求的前缀 T_{pref}。

选择高度上限为 n^{ε} 并非最佳方案，这并不令人意外。具体而言，若将初始高度选为 $2^{\mathcal{O}_k(\sqrt{\log n} \log \log n)}$，则上述相同策略仍然适用，从而表明我们可以将每次查询的摊还运行时间降至 $2^{\mathcal{O}_k(\sqrt{\log n} \log \log n)}$。

3 预备知识

对于任意正整数 n，我们定义 $[n] \coloneqq \{1, 2, \ldots, n\}$。给定一组参数 $\bar{c}$，记号$\mathcal{O}_{\bar{c}}(\cdot)$ 隐去一个乘法因子，该因子有一个可计算函数上界，且该函数依赖于 $\bar{c}$。

图。我们使用标准的图论记号。本文仅考虑无向简单图。对于图 G，我们分别用 $V(G)$ 和 $E(G)$ 表示 G 的顶点集和边集。G 的连通分量集合记作 $\mathsf{cc}(G)$。对于集合 $A \subseteq V(G)$，我们用 $G[A]$ 表示 G 中由 A 诱导的子图，并用 $G - A$ 表示 G 中由 $V(G) \setminus A$ 诱导的子图。

给定顶点 $v \in V(G)$，我们用 $N_G(v)$ 表示其（开）邻域，用 $N_G[v] \coloneqq N_G(v) \cup \{v\}$ 表示其闭邻域。这一记号可扩展到 G 的顶点子集的（开和闭）邻域：给定 $A \subseteq V(G)$，定义 $N_G[A] \coloneqq \bigcup_{v \in A} N_G[v]$ 和 $N_G(A) \coloneqq N_G[A] \setminus A$。当上下文已明确图 G 时，我们可以在这一记号中省略 G。

对于集合 $A \subseteq V(G)$，我们将 A 在 G中的**躯干图** 定义为 $\mathsf{torso}_G(A)$，即顶点集为 A 且满足以下条件的图 H：$uv \in E(H)$ 当且仅当 $u,\ v \in A$ 并且存在一条连接 u 和v 的路径，该路径与 A内部不相交。等价地，$u,\ v \in A$ 并且满足 $uv \in E(G)$，或者存在一个连通分量 $C \in \mathsf{cc}(G - A)$ 使得 $\{u, v\} \subseteq N_G(C)$。

给定两个集合 $A, B \subseteq V(G)$，如果集合 $S \subseteq V(G)$ 满足：连接 A 中的一个顶点与 B 中的一个顶点的每条路径都与 S相交，则称其为 (A, B)-分隔集。如果集合 A, S, B 构成 $V(G)$ 的一个划分，则 (A, S, B) 是 G 的一个分离。该分离的阶为 $|S|$。接下来，如果每个 (A, B)-分隔集 S 的规模至少为 $|A|$，则称 A 与 B 相连。等价地（根据门格尔定理），存在 $|A|$ 条连接 A 和 B 的顶点不相交路径。注意，如果 $B \subseteq B'$ 和 A 与 B 相连，则A也与 B' 相连。

树。我们通常将树的顶点称为**节点**，以便与图的顶点区分。除非另有明确说明，本文中的所有树都是**有根的**：树中恰有一个节点被指定为树的根。这自然地确立了树中的父子关系和祖先/后代关系。这里，我们假设每个节点既是自身的后代，也是自身的祖先。如果每个节点至多有两个子节点，我们就称这棵树为**二叉** 树。本文中，我们不区分二叉树节点的左子节点和右子节点。

节点 v 的**深度** 是其到根的距离，而节点 v 的**高度** 是其到后代 v 的最大距离再加一。特别地，根的深度为 0，而任何叶节点的高度均为 1。

（有根）**森林** 是由若干棵互不相交的（有根）树组成的集合。如果两个不同的节点有共同的父节点，或者它们都是根，则称它们为**兄弟节点**。（后一种情况只可能出现在有根森林中。）

在一棵树 T 中，如果集合 $W \subseteq V(T)$ 满足：对每个非根节点 $t \in W$，其父节点 $\mathsf{parent}(t)$ 也属于 W，则称 $W \subseteq V(T)$ 是 T 的 **前缀**。如果节点 t 不属于 W，但其父节点属于该集合，则称 t 是 W 的一个 **附属节点**。等价地，$N_T(W)$ 是 W 的所有附属节点构成的集合，也记作 $\mathsf{App}(W)$。下面的事实显然成立：

事实 3.1. 如果 T 是一棵二叉树，且 W 是 T 的前缀，那么 W 至多有 $|W| + 1$ 个附属节点。

两个节点 $s,\ t$ 的**最近公共祖先** 概念，记作 $\mathsf{lca}(s,\ t)$，按标准方式定义。我们称 T 的节点子集 $X \subseteq V(T)$ 是**最近公共祖先闭合** 的，若满足以下性质：对任意两个节点 $s,\ t \in X$，节点 $\mathsf{lca}(s,\ t)$ 也属于 X。给定集合 $A \subseteq V(T)$，我们将 A 的**最近公共祖先闭包** 定义为包含意义下唯一的极小最近公共祖先闭合集 $X \supseteq A$。等价地，$X = \{\mathsf{lca}(s,\ t) \colon\ s,\ t \in A\}$。以下事实是标准结论：

事实 3.2。 $A \subseteq V$（T）的最近公共祖先闭包 X 满足 $|X| \leqslant 2|A| - 1$。

事实 3.3。 如果 X 是 $A \subseteq V(T)$的最近公共祖先闭包，那么 X 的每个节点都是 A中某个节点的祖先。

树分解。 图G的树分解是一个二元组（T，bag_T），由一棵树T和一个函数 bag_T：$V(T) \to 2^{V(G)}$组成，其中：

- 对于每个顶点$v \in V(G)$，节点子集 $\{t \in V(T) : v \in \mathsf{bag}_T(t)\}$所诱导的是一个非空连通子树 T（顶点条件）；并且
- 对于每条边 $uv \in E(G)$，都存在一个节点 $t \in V(T)$，使得 $\{u, v\} \subseteq \mathsf{bag}_T(t)$（边条件）。

分解（T，bag_T）的宽度 为 $\max_{t \in V(T)} |\mathsf{bag}_T(t)| - 1$。 G的树宽 记作 $\mathsf{tw}(G)$，是 G的任意树分解可能达到的最小宽度。

本文中，所有树分解都是有根的；也就是说， T始终是一棵有根树。如果T是一棵（二叉）树，那么我们称 $(T, 袋_T)$ 为二叉树分解。

我们还引入以下简写：给定T中节点的一个子集 $W \subseteq V(T)$，我们令 $袋_T(W) := \bigcup_{t \in W} 袋_T(t)$。通常可以从上下文中得知树 T，因此我们将省略下标，分别用袋和袋代替$袋_T$ 和$袋_T$。此外，稍微滥用一下记号，当树分解 $(T, 袋)$ 可由上下文明确时，我们可以简单地用 $\mathcal{T}$作为 $(T, 袋)$ 的简写。

非根节点 $t \in V(T)$的粘合集定义为 $\mathsf{adh}(t) := 袋(t) \cap 袋(\mathsf{parent}(t))$；如果 t 是 T的根，则令 $\mathsf{adh}(t) := \emptyset$。接着， t 的连通分量定义如下：

$$\mathtt{cmp}(t) := \left(\bigcup \{\mathtt{bag}(s) : s \text{ is a descendant of } t \text{ in } T\}\right) \setminus \mathtt{adh}(t).$$

我们注意到，$(\mathsf{cmp}(t), \mathsf{adh}(t) \ V(G) \setminus (\mathsf{cmp}(t) \cup \mathsf{adh}(t)))$ 是 G 的一个分离。

我们指出下面这个众所周知的事实：

事实 3.4。 如果 A 是 G 中的一个团，且 $\mathcal{T}$ 是 G 的一个树分解，那么 $\mathcal{T}$ 的某个袋必须包含 A。

最后，对于图 G 的树分解 $\mathcal{T} = (T, \mathsf{bag})$，我们定义深度函数

$$d_{\mathcal{T}} : V(G) \to \mathbb{Z}_{\geqslant 0}$$

该函数将每个顶点 $v \in V(G)$ 映射到满足 $v \in \mathsf{bag}(t)$ 的最浅节点 $t \in V(T)$ 的深度。如果上下文中已知树分解 $\mathcal{T}$，我们将写作 d，而不是 $d_{\mathcal{T}}$。

动态树分解。 现在，我们介绍一种通用的数据结构设计，它将在动态变化的图的动态变化的二叉树分解上运行。

将图 G 的带注释的树分解 定义为三元组 $(T, 袋, 边)$，其中：

- T和袋的定义与标准树分解定义中的相同；
- 边：$V(T) \to 2^{\binom{V(G)}{2}}$定义如下：对于节点 $t \in V(T)$，$\mathsf{edges}(t)$ 是 $\binom{\mathsf{bag}(t)}{2}$的一个子集，由所有满足以下条件的边 $uv \in E(G) \cap \binom{\mathsf{bag}(t)}{2}$组成：$t$是同时包含 u和 v的最浅节点。注意，因此， G 的每条边恰好属于一个集合 $\mathsf{edges}(t)$。

注意，edges 由 G 和 (T,bag,G) 唯一确定；反之，G 由 $(T,\mathsf{bag},\mathsf{edges})$ 唯一确定。给定一个集合 $A\subseteq V(T)$，将 $(T,\mathsf{bag},\mathsf{edges})$ 限制到 A，记为 $(T,\mathsf{bag},\mathsf{edges})|_A$，它是元组 $(T[A],\mathsf{bag}|_A,\mathsf{edges}|_A)$，其中 $\mathsf{bag}|_A$ 和 $\mathsf{edges}|_A$ 分别是函数 bag 、 edges 在 A 上的限制。

接下来，考虑一次更新，它将带注释的二叉树分解 $(T,$ 袋, 边$)$ 改为另一个带注释的二叉树分解 $(T',$ 袋′, 边′$)$。这次更新也可能改变底层图 G；具体而言，它会将 G 改为由 $(T',$ 袋′, 边′$)$ 唯一确定的图。我们称这次更新为前缀重建，当且仅当 $(T',$ 袋′, 边′$)$ 是通过将 $(T,$ 袋, 边$)$ 中的一个前缀 T_{pref} 替换为一棵新的有根树 T'_{pref}，然后将以 T_{pref} 的附属子树为根的 T 中的一些子树“重新连接”到 T'pref 的节点下方而创建的。形式上，前缀重建更新由一个元组 $u:=(T_{\mathrm{pref}},T'\text{pref}\ T^\star,$ 袋*, 边$^\star,\pi)$ 描述，其中：

- $T_{\text{前缀}}\subseteq V(T)$ 是 T 的前缀；
- $T'_{\text{前缀}}\subseteq V(T')$ 是满足以下条件的 T' 的前缀

$$(T,\mathsf{bag},\mathsf{edges})|_{V(T)\setminus T_{\mathrm{pref}}}=(T',\mathsf{bag}',\mathsf{edges}')|_{V(T')\setminus T'_{\mathrm{pref}}};$$

- $(T^\star,$ 袋*, 边$^\star)=(T',$ 袋′, 边′$)|_{T'\text{前缀}}$；
- π：$\mathrm{App}(T_{\text{前缀}})\rightharpoonup T'_{\text{前缀}}$ 是一个部分函数，它将 $T_{\text{前缀}}$ 的附属节点映射到 $T'_{\text{前缀}}$ 的节点，使得对于 $T_{\text{前缀}}$ 的每个附属节点 t，若 $\pi(t)$ 有定义，则 T' 中 t 的父节点为 $\pi(t)$。

显然，可以根据上文的元组 u，唯一确定 $(T',$ 袋′, 边′$)$。u 的规模 记为 $|u|$，定义为 $|T_{\mathrm{pref}}|+|T'_{\text{前缀}}|$。同样显然，给定 u，可以在 $\ell^{\mathcal{O}(1)}\cdot|u|$ 时间内将 $(T,$ 袋, 边$)$ 的一种表示转换为 $(T',$ 袋′, 边′$)$ 的一种表示，其中 ℓ 是 $(T,$ 袋, 边$)$ 和 $(T',$ 袋′, 边′$)$ 的宽度最大值。

最后，如果一个动态数据结构存储了宽度至多为 ℓ 的带注释的二叉树分解（T、袋、边），并支持以下操作，则称其为具有 ℓ-前缀重建开销 τ 的动态数据结构：

- 初始化$(T$、袋、边$)$：使用（T、袋、边）初始化带注释的二叉树分解。最坏情况时间为 $\mathcal{O}(\tau\cdot|V(T)|\cdot\ell^{\mathcal{O}(1)})$；
- 更新(u)：对分解（T、袋、边）应用前缀重建更新 u。可以假设所得树分解为二叉树分解，且宽度至多为 ℓ。最坏情况时间为 $\mathcal{O}(\tau\cdot|u|\cdot\ell^{\mathcal{O}(1)})$。

通常，开销 τ 对应于重新计算分解中因更新而受影响的每个节点所关联的辅助信息所需的时间。例如，树分解中节点 s 的高度可以根据 s 的（至多两个）子节点的高度在 $\mathcal{O}(1)$ 时间内推断出来，因此，更新后重新计算节点高度所需的开销为每个受影响节点 $\tau=\mathcal{O}(1)$。

前缀重建数据结构通常还会实现一种额外操作，以便高效查询数据结构的当前状态。例如，下面我们给出一种数据结构，可用于访问有关树分解的各种辅助信息：

引理 3.5。对于每个 $\ell\in\mathbb{N}$，都存在一个开销为 $\mathcal{O}(1)$的 ℓ-前缀重建数据结构，并且该结构还实现以下操作：

- 高度(s)：给定一个节点 $s \in V(T)$，返回 s 在 T 中的高度。最坏情况时间为 $\mathcal{O}(1)$。
- 规模(s)：给定一个节点 $s \in V(T)$，返回以 s 为根的 T 子树中的节点数。最坏情况时间为 $\mathcal{O}(1)$。
- 连通分量大小(s)：给定一个节点 $s \in V(T)$，返回规模为 $|\texttt{cmp}(s)|$。最坏情况时间为 $\mathcal{O}(1)$。
- 顶(v)：给定一个顶点 $v \in V(G)$，返回 T 的唯一最高节点 t，使得 $v \in$ 袋(t)。最坏情况时间为 $\mathcal{O}(1)$。

引理 3.5 的引理 3.5 证明使用了树分解上动态规划的标准论证。证明将在附录 A中给出。

更一般地，树分解上的任何典型动态规划方案都可以转化为前缀重建数据结构。下面给出一个目前非正式表述的结论。

引理3.6（非正式）。固定 $\ell \in \mathbb{N}$。假设存在一个动态规划方案，作用于宽度至多为 ℓ 的二叉树分解，其中树分解的节点 t 的状态仅取决于 袋（t）、边（t） 以及树中 t 的子节点的状态；并且该状态可以根据这些信息在时间 τ 内计算得到。那么，存在一个开销为 τ 的 ℓ-前缀重建数据结构，该数据结构还实现以下操作：

- 状态(s)：给定一个节点 $s \in V(T)$，返回该节点的状态 s。最坏情况时间为 $\mathcal{O}(1$).

在附录 A中，我们通过合适的自动机模型形式化说明树分解上的“动态规划方案”所指为何。随后，引理 3.6被形式化为一条陈述（引理 A.6）：该陈述指出，可以在前缀重建更新下维护自动机在树分解上的运行，而引理 3.5的前三个要点则通过将该陈述应用于特定（非常简单的）自动机得到了形式化证明。在下文的若干处，我们需要在前缀重建更新下维护树分解上更复杂的动态规划方案。每种情况下，我们都会陈述一个关于前缀重建数据结构存在性的适当引理，并在附录 A中使用合适的自动机构造来证明该引理。

最后，我们证明，对于不改变底层图 G 的前缀重建更新，可以取消“更新描述中给出函数 edges* ”这一假设。考虑一个不改变图 G 的前缀重建更新，并将该更新的一个弱描述 定义为一个元组 $\widehat{u} := (T_{\text{pref}}, T'_{\text{pref}} T^\star, \text{袋}^\star, \pi)$；该元组除 edges 函数外，须满足前缀重建更新描述的相同性质。由于图 G 未发生变化，新的带注释的二叉树分解（T', 袋′, 边′）可以由（T, 袋, 边）和 $\widehat{u}$ 唯一确定。我们仍将 $|\widehat{u}| = |T_{\text{pref}}| + |T'_{\text{pref}}|$ 记作。

我们证明，前缀重建更新的弱描述 $\widehat{u}$ 可以转换为前缀重建更新的描述u，使得 $|u| = \mathcal{O}(|\widehat{u}|)$ 以及应用 u 后得到的带注释的二叉树分解 $(T'$，袋′，边′$)$ 与应用 $\widehat{u}$后得到的分解相同。我们注意到，该操作可能会使集合 T_{pref} 和 T'_{pref} 变大，但其增幅由 $\mathcal{O}(|\widehat{u}|)$ 界定。

引理 3.7。对于每个 $\ell \in \mathbb{N}$，存在一个具有开销 $\mathcal{O}(1)$的 ℓ-前缀重建数据结构，该数据结构还实现以下操作：

- 强化$(\widehat{u})$：给定一个前缀重建操作的弱描述 $\widehat{u}$，返回一个前缀重建操作的描述u，使得 $|u| = \mathcal{O}(\widehat{u})$，且应用 $\widehat{u}$ 和 u 会得到相同的带注释的树分解$(T'$, 袋′，边′$)$。最坏情况运行时间为 $|\widehat{u}| \cdot \ell^{\mathcal{O}(1)}$。

证明。令 $\widehat{u} = (\widehat{T}_{\text{pref}}, \widehat{T}'$前缀 $\widehat{T}^\star$, $\widehat{\text{袋}}^\star, \widehat{\pi})$和$(T'$，袋′，边′)为所得的带注释树分解。我们注意到，包含边 $uv \in E(G)$的最高层袋，只有在$\widehat{\quad}$ $\widehat{\quad}$同时满足 u、$v \in$袋$_{T'}(T'$前缀)时才会改变。然而，如果 u、 $v \in$袋$_{T'}(T'$前缀)，那么根据$(T'$，袋′)的顶点条件，必有 u、 $v \in$袋$_T(T_{\text{pref}} \cup \text{App}(T_{\text{pref}}))$；特别是，$(T$，袋)中的顶点条件意味着 uv必须存储在边$(T_{\text{pref}} \cup \text{App}(T_{\text{pref}}))$中。因此，对边函数的更改$\widehat{\quad}$ $\widehat{\quad}$仅限于由 $T_{\text{pref}} \cup \text{App}(T_{\text{pref}})$构成的子树 T。因此，要构造$u = (T_{\text{pref}}, T'$前缀 $T^\star$，袋*，边$^\star, \pi)$，只需取 $T_{\text{pref}} = T_{\text{pref}} \cup \text{App}(T_{\text{pref}})$，并以类似方式从 $\widehat{T}'$前缀 $\widehat{T}^\star$、 $\widehat{\text{袋}}^\star$和 $\widehat{\pi}$构造 T'前缀、 $T^\star$和袋*，以及 π。然后，可以根据$(T^\star$，袋$^\star)$以及限制在 $T_{\text{pref}} \cup \text{App}(T_{\text{pref}})$上的边函数，确定边$^{\star\widehat{\quad}}$ $\widehat{\quad}$函数。运行时间和 $|u|$ 的界由树分解是二叉的这一事实得出。

□

现在，通过使用引理 3.7中的数据结构，我们可以假设，在构造不改变 G的前缀重建操作时，只需构造弱描述；但在实现前缀重建数据结构时，更新方法接收的是一个（非弱）描述。在本文余下部分，我们假设始终维护着引理 3.7中的数据结构；具体来说，通常先用它将弱描述 $\widehat{u}$ 转换为描述u，然后立即对其应用更新(u)。

逻辑。我们使用 CMSO_2 ——一种用于图的单子二阶逻辑，允许对边子集进行量化并包含模计数谓词——它通常与有界树宽图相关；参见 [24, 第 7 节。4]，其中介绍了面向算法设计者的相关内容。CMSO_2 公式在图上求值，并包含四种不同类型的变量：分别对应单个顶点、单条边、顶点子集和边子集。后两种类型称为单子。CMSO_2 的原子公式形式如下：

- 相等： $x = y$，其中 x， y 均为单个顶点变量或边变量。
- 隶属关系： $x \in X$，其中 x 是单个顶点变量或边变量， X 是单子顶点变量或边变量。

- 关联关系：$\text{inc}(x, e)$，其中 x 是单个顶点变量， e 是单个边变量。
- 模计数： $|X| \equiv a \pmod m$，其中 a和 m 是整数， $m > 0$。

上述内容的语义符合预期。接着，CMSO_2 由所有可通过以下构造从原子公式得到的公式组成：标准布尔联结词、否定，以及对所有类型变量进行量化，包括存在量化和全称量化。因此，CMSO_2公式可能包含未受任何量词约束的变量；这类变量称为自由变量。不含自由变量的公式称为语句。对于语句 φ 和图 G，我们用 $G \models \varphi$ 表示φ 在 G 中成立（读作 G是.....的模型 φ）。

Courcelle 定理指出，给定一个图 G ，其树宽为 k ，以及一个CMSO_2 公式 φ，可以在 $G \models \varphi$时间内判定 $f(k, \varphi) \cdot n$，其中 n是 G的顶点数，而 f是一个可计算函数。在 Courcelle 定理的证明中，通常首先计算 G 的树分解，其宽度至多为 k，例如使用 Bodlaender 的算法 [13]，，然后应用一个从 φ 中适当构造的动态规划过程（即自动机）来验证 φ是否满足。我们证明，这一动态规划过程可以在前缀重建更新下得到维护。更正式地，在附录 A 中，我们证明以下结论。

引理 3.8。 固定 $\ell \in \mathbb{N}$ 和一个 CMSO_2 语句 φ。那么存在一个开销为 $\mathcal{O}_{\ell,\, \varphi}(1)$的 ℓ-前缀重建数据结构，并且它还实现以下操作：

- 查询()：返回是否 $G \models \varphi$。最坏情况时间为 $\mathcal{O}_{\ell,\varphi}(1)$。

在树宽较小这一承诺下的动态树分解。 所有定义就位后，我们终于可以陈述用于证明 定理 1.1 的核心结果。

引理 3.9. 存在一种数据结构，对于一个在初始化时固定的整数 $k \in \mathbb{N}$，以及一个通过边插入和删除进行更新的动态图 G，维护一个带注释的树分解$(T$，袋，边$)$，其对应于 G，宽度至多为 $6k+5$，并在以下承诺条件下使用前缀重建更新：$\mathrm{tw}(G) \leqslant k$ 始终成立。更准确地说，在任意时刻，都保证该图的树宽至多为 k，且该数据结构包含一个对应于 G、宽度至多为 $6k+5$的带注释树分解。该数据结构可在时间 $2^{\mathcal{O}(k^8)} \cdot n$内，以 k 和一个无边的 n 顶点图 G 为输入完成初始化，之后每次更新：

- 返回用于修改该树分解的前缀重建更新序列；并且
- 的摊销时间为 $2^{\mathcal{O}\left(k^9 + k \text{ 对数 } k \cdot \sqrt{\text{对数 } n \text{对数对数 } n}\right)}$。

注意，根据引理 3.9可直接得出，数据结构在前 q 次边插入/删除操作中返回的所有前缀重建更新的总规模有界，为

$$2^{\mathcal{O}(k^8)} \cdot n \; + \; 2^{\mathcal{O}(k^9 + k \log k \cdot \sqrt{\log n \log\log n})} \cdot q.$$

引理 3.9 在第 7 节 中利用第 4 节至第 6 节的结果加以证明。值得注意的是，该引理的表述本质上是定理 1.1的一个较弱版本：首先，我们假设不会有任何使$\mathrm{tw}(G)$ 增至 k 以上的更新进入数据结构；其次，我们不支持动态CMSO_2 模型检测。我们在附录 B 中分别通过以下方法解决这些问题：直接应用 Eppstein 等人提出的延迟破坏不变量的插入技术 [29],以及引理 3.8。

4 闭包

本节介绍一种图论概念——闭包，它将在本文后文对算法的介绍中用到。本节余下部分中，固定一个整数 $k \in \mathbb{N}$，并令G为一张树宽至多为 k的图。

直观地说，当人们尝试动态维护一个图的树分解时，不可避免地会遇到这样的情况：所维护的树分解中的某些袋过大。考虑以下改进这种树分解的朴素方法：令 W 为被认为过大的袋的并集。构造一个躯干图$_G(W)$的（有根）树分解 T_W。然后，对于每个连通分量 $C \in \mathrm{cc}(G - W)$，集合 $N(C)$是躯干图$_G(W)$中的一个团；因此，整个集合 $N(C)$都包含在 T_W的单个袋 t_C中。因此，可以通过构造一个根袋完全包含$N(C)$的 $G[N[C]]$ 的树分解 T_C，再将 T_C 的根连接到袋 $t_C \in V(T_W)$，把 C并入 T_W。可以直接验证，这是一个有效的 G的树分解构造。

然而，目前尚不清楚这种构造为何能改善所维护分解的宽度。这是因为躯干图$_G(W)$的树宽原则上可能远大于 k。不过，人们或许可以希望，集合 W 能被一个仅稍大的集合 $X \supseteq W$ 覆盖，使得躯干图$_G(X)$的树宽很小。事实确实如此，由此引出 W的闭包 的定义：

定义 1. 设 $k \in \mathbb{N}$ 和 G 满足$\mathrm{tw}(G) \leqslant k$。另设 $W \subseteq V(G)$。如果满足以下条件，则称集合 $X \subseteq V(G)$ 为 W 在 G 中的 k-闭包：

$$X \supseteq W \qquad \textit{and} \qquad \mathrm{tw}(\mathtt{torso}_G(X)) \leqslant 2k+1.$$

注意，每个集合 W 都有一个平凡闭包 $X = V(G)$，因为 $\mathrm{torso}_G(V(G)) = G$。显然，这样的闭包可能比 W 大得多。幸运的是，下面的引理说明了如何构造规模更易于处理的闭包：

引理 4.1. 设 $(T,$袋$)$ 是 G 的一个宽度至多为 k 的树分解。再设 S 是 T 的节点的一个最近公共祖先封闭集。那么，

$$\mathrm{tw}(\mathtt{torso}_G(\mathtt{bags}(S))) \leqslant 2k+1.$$

证明. 我们按以下方式构造一棵有根树 U：令 $V(U) = S$，并且当且仅当 t_1 是 t_2 的严格祖先，且 t_1 与 t_2 在 T 中的简单路径不包含 S 中的其他顶点时，令 t_1 为 t_2 在 U 中的父节点。由于 S 是最近公共祖先封闭的，可以很容易地验证 U 确实是一棵有根树。我们还按如下方式构造树分解 $(U,$袋′$)$：

$$\mathtt{bag}'(t) = \begin{cases} \mathtt{bag}(t) & \text{if } t \text{ is the root of } U, \\ \mathtt{bag}(t) \cup \mathtt{bag}(\mathtt{parent}_U(t)) & \text{otherwise.} \end{cases}$$

我们断言，$(U,$袋′$)$ 是 $\mathrm{torso}_G($袋$(S))$ 的一个宽度至多为 $2k+1$ 的树分解。顶点条件很容易验证。对于边条件，考虑 $\mathrm{torso}_G($袋$(S))$ 的一条边 uv。我们有 $u, v \in$ 袋(S)，并且在 G 中存在一条连接 u 与 v 的简单路径 P，该路径与袋(S) 内部不相交。选取 S 的两个节点 t_u、t_v，使得 $u \in$ 袋(t_u)，$v \in$ 袋(t_v)。对于 t_u 与 t_v 在 U 中路径上的每个节点 t，集合袋(t) 必须包含 u 或 v；否则，袋 (t) 就会是在 G 中分隔 u 与 v 、且与 $\{u, v\}$ 不相交的分隔集，这与 P 的存在相矛盾。因此，以下情况之一必然成立：

- u 和 v 都属于某个 $t \in S$的袋(t)。于是 u，$v \in$ 袋′(t)，因此边 uv满足边条件。

- 对于某些节点 t_1，$t_2 \in S$，我们有 $u \in$ 袋(t_1)、$v \in$ 袋(t_2)，且这些节点在 U中相邻。不失一般性，假设 t_1是 t_2的父节点。于是，由于袋′(t_2) =袋$(t_1)\cup$袋(t_2)，我们推得 u，$v \in$ 袋′(t_2)。

我们得出结论，$(U$，袋′$)$确实是躯干图$_G($bags$(S))$的树分解。由于$(U$，袋′$)$的每个袋的规模至多为 $2k+2$，证明完成。 □

引理 4.1 已经表明，每个集合 $W \subseteq V(G)$都存在一个基数至多为$\mathcal{O}(k) \cdot |W|$的闭包 X。事实上，考虑一个宽度最小的 G 的树分解$(T_{\mathrm{opt}},$ 袋$)$。对于每个顶点 $v \in W$，在 S 中选取一个节点 $t \in V(T_{\mathrm{opt}})$，使得 $v \in$ 袋(t)。然后，取S 的最近公共祖先闭包（这会使 $|S|$至多增加一倍），并应用引理 4.1。

遗憾的是，这在我们的情形下并不足够。在我们的算法中，当我们维护树分解 $(T,$ 袋$)$时，集合 W将被选为前缀 T_{pref}中的袋的并集，该前缀属于 T。在这种设置下，还需要对闭包 $X \supseteq W$ 提出另一项条件：对于 t 中的每个附属节点 T_{pref}，整个连通分量 $\mathtt{cmp}(t)$ 中属于 X 的顶点数量必须有界。这样的闭包称为小闭包。我们将在第 4.1 节中的引理 4.2（小闭包引理）中证明此类闭包的存在性。随后，我们将在第 4.2 节证明一个结构性结果。

关于小闭包（引理 4.7，闭包连通性引理）。接下来，在第 4.3 节中，我们将定义与闭包相关、且对树分解改进算法至关重要的对象——阻塞集、探索集和收集的连通分量——并证明这些概念的若干结构性质。最后，第 4.4 节将概述如何在动态变化的树分解中高效地寻找小闭包。

4.1 小闭包引理

现在我们正式定义小闭包的概念。

定义 2。 设$(T,$ 袋$)$ 是 G 的树分解，且 T_{pref} 是 T 的前缀。另设 $c \in \mathbb{N}$ 为一个整数。若对于 T_{pref} 的每个附属节点 t，都有 $|X \cap \mathtt{cmp}(t)| \leqslant c$，则称集合 $X \supseteq$ 袋(T_{pref}) 关于$(T,$ 袋$)$ 是 c-小的 。

有了这个定义，我们现在可以陈述小闭包引理了。该引理断言，当 c 足够大时，存在 c-小闭包：

引理 *4.2*（小闭包引理）。存在一个函数 $g(\ell) \in \mathcal{O}(\ell^4)$，使得以下结论成立。设 k、$\ell \in \mathbb{N}$ 满足 $k \leqslant \ell$，且 G 是一个图，其 $tw(G) \leqslant k$。设 $(T,$ *bag*$)$ 是 G 的一个宽度至多为 ℓ 的树分解，且 $T_{\mathrm{pref}} \subseteq V(T)$ 是 T 的前缀。那么，存在一个关于 $(T,$ *bag*$)$ 的 $g(\ell)$-小 k-闭包 $bags(T_{\mathrm{pref}})$。

证明 引理 4.2 使用 Bojańczyk 和 Pilipczuk 的方法 [17]，即去交替引理：直观地说，由于 T 是 G的有界宽度分解，U 因此存在一个良结构树分解 G，使得对于每个附属节点 t 所属的 T_{pref}连通分量$\mathtt{cmp}(t)$ 都可以划分为数量有界的、关于 U的良结构“块”。闭包 X的构造方式将保证每个块 U 只包含来自 X的有界数量的顶点。因此， X 将从每个$\mathtt{cmp}(t)$中包含有界数量的顶点。

现在我们给出去交替引理的正式版本。描述沿用 [17], 的讲解方式，并省略了对我们无关的细节。

消去森林。 去交替引理的输出是对 G 的一个树分解，并以所谓的消去森林形式呈现：

定义 3。 G 的一个消去森林是定义在顶点集 $V(G)$ 上的有根森林 F，满足以下性质：若 $uv \in E(G)$，则 u 和 v 在 F 中具有祖先-后代关系。

下面的定义说明如何将 G 的消去森林转化为树分解：

定义 4. 假设 F 是 G 的消去森林（因此 $V(F) = V(G)$）。由 F诱导的树分解是树分解 $(F,$袋$)$，其中对于每个 $v \in V(G)$，我们令 袋(v) 包含 v，以及通过 G 的一条边与 v 的任一后代相连的 v 的每个祖先。

在上述定义中，我们稍微滥用了记号，允许树分解的形状为有根森林，而非有根树；其他所有条件保持不变。请注意，这样的森林分解总能通过选取一个根 r，并将其他所有根设为 r 的子节点，转化为宽度相同的树分解。

该$(F,$ 袋$)$ 按定义 4构造，确实是 G，其论证见[17, 第 3]节。现在自然可以定义消去森林的宽度 F为由 F诱导的树分解的宽度。显然，每个消去森林的宽度至少为$\mathrm{tw}(G)$。另一方面，每个图 G 都有一个宽度恰好为$\mathrm{tw}(G)$ [17, 引理 36]。

因子。直观而言，因子是森林 F 中结构良好的“块”。形式上，因子是 $V(F)$ 的一个子集，且满足以下条件之一：

- 一个森林因子： F的非空有根子树集合的并集，这些子树的根彼此都是兄弟节点；或者
- 一个上下文因子：形如 $\Phi_1 \setminus \Phi_2$ 的非空集合，其中 Φ_1 是一棵有根子树，且 $\Phi_2 \subseteq \Phi_1$ 是一个森林因子；上下文因子的根是 Φ_1 的根，而 Φ_2 中树因子的根称为附属子树。

去交替引理。 现在我们可以陈述去交替引理。

引理 *4.3*（去交替引理， [17]） 。存在一个函数 $f(\ell) \in \mathcal{O}(\ell^3)$，使得以下结论成立。设 *(T,* 袋 *)* 是 G 的一个宽度至多为 ℓ 的树分解。那么存在 G 的一个宽度为 *tw(G)* 的消去森林 F，使得对于每个节点 $t \in V(T)$，集合 *cmp(t)* 都是 F 的至多 $f(\ell)$ 个因子的互不相交并集。

小闭包引理的证明。 现在我们展示去交替引理如何推出小闭包引理(引理 4.2)。

引理 4.2 的证明。回顾一下，已知：一个树宽至多为 k 的图 G；一个宽度至多为 ℓ 的 G 的树分解 $(T, \text{袋})$，其中 $\ell \geqslant k$；以及 T 的一个前缀 T_{pref}。我们需要找到袋(T_{pref}) 的一个小 k-闭包 X。我们强调，这要求 $\text{tw}(\text{躯干图}_G(X)) \leqslant 2k+1$。

我们首先将引理 4.3 应用于 $(T, \text{袋})$，并得到 G 的一个宽度为 $\text{tw}(G)$ 的消去森林 F，使得对于 $t \in V(T)$，每个 $\text{cmp}(t)$ 都可以分解为至多 $f(\ell)$ 个 F 的 $\in \mathcal{O}(\ell^3)$ 因子。我们指出，$V(F) = V(G)$。

如定义 4 后的段落所述，存在一个宽度为 $\text{tw}(G)$ 的 G 的树分解（F，bag'），其中，对于 $v \in V$（G），$\text{bag}'(v)$ 定义为包含 v 以及 F 中与一条边关联的、且该边另一端点是 v 的后代的 v 的每个祖先的集合。令 $W \coloneqq \text{bags}_T(T_{\text{pref}})$，并令 W' 为 F 中 W 的最近公共祖先闭包。我们声称集合

$$X \coloneqq \bigcup_{v \in W'} \text{bag}'(v)$$

is an $((\ell+1)\cdot f(\ell))$-小 k 闭包（相对于 T 的 W）。由此即可完成证明 f.

命题 4.4。 X 是 $\text{bags}_T(T_{\text{pref}})$ 的 k 闭包。

命题证明。由于 $\text{bags}_T(T_{\text{pref}}) \subseteq W'$，并且对每个 $v \in W'$，$v \in \text{bag}'(v)$，我们有 $\text{bags}_T(T_{\text{pref}}) \subseteq X$，正如所需。剩下要证明 $\text{tw}(\text{torso}_G(X)) \leqslant 2k+1$。然而，由于 W' 在 F 中是最近公共祖先闭合的，引理 4.1 适用于 W' 以及树分解（F，bag'）的（每个）连通分量，从而完成证明。◁

命题 4.5. 设 t 是 T_{pref} 的附属节点，且 Φ 是 F 的因子，其中 $\Phi \subseteq \text{cmp}(t)$。那么

$$|\Phi \cap X| \leqslant \ell + 1.$$

命题证明。 由于 t 是 T_{pref}的附属节点，根据 $\text{cmp}(t)$ 的定义，$\text{cmp}(t)$ 与袋$_T(T_{\text{pref}})$ 不相交。因此，Φ也与袋$_T(T_{\text{pref}})$ 不相交。

首先，假设 Φ 是一个森林因子。那么 Φ 是向下封闭的：如果一个顶点属于 Φ，那么它的所有后代也都属于 Φ。由于 W' 由袋 $_T(T_{\text{pref}})$ 以及其祖先的一个子集组成

袋 $T(T_{\text{pref}})$，因此可知 Φ 与 W' 不相交。现在，对于每个 $v \in W'$，集合袋 $'(v)$ 包含 v 以及 v 的一些祖先。因此， Φ 与每个满足 $v \in W'$ 的集合袋 $'(v)$ 均不相交，进而与 X 不相交。

现在假设 Φ 是一个上下文因子。回忆一下， $\Phi = \Phi_1 \setminus \Phi_2$，其中 Φ_1 是 F中以某个顶点 r为根的子树，而 $\Phi_2 \subseteq \Phi_1$ 是一个森林因子。 Φ 的附属子树有一个共同的父节点，我们称之为 s。由袋$T(T_{\text{pref}})$与 Φ不相交可知，袋$T(T_{\text{pref}})$中的每个顶点要么位于 Φ_1 之外，要么位于 Φ_2的某个子树内。

由于 W'是袋$T(T_{\text{pref}})$的最近公共祖先闭包，因此我们有 $W' = \{\text{lca}(u,\ v)\colon\ \ u,\ \ v \in$袋$T($ $T_{\text{pref}})\}$。考虑满足$\text{lca}(u,\ v)\ \in \Phi_1$的 $u,\ \ v \in$袋$T(T_{\text{pref}})$。如果 u 或 v 位于 Φ_1之外，那么$\text{lca}(u,\ v)$也位于 Φ_1之外。因此， u 和 v 都属于 Φ_2。在这种情况下，很容易看出，$\text{lca}(u,\ v)$要么属于 Φ_2 （如果 u 和 v 来自 Φ_2的同一棵有根树），要么等于 s（否则）。因此，

$$\Phi_1 \cap W' \subseteq \Phi_2 \cup \{s\}.$$

请注意， $\Phi_2 \cup \{s\}$ 是 F 的一个连通子图，包含 s 以及附着在 s上的一些有根子树。

同样，对于每个 $v \in W'$，袋$'(v)$由 v 和 v的一些祖先组成。因此，对于 $v \in W' \setminus \Phi_1$，袋$'(v)$与 Φ不相交。因此，

$$\Phi \cap X \subseteq \bigcup\{\Phi \cap \mathtt{bag}'(v)\colon\ v \in \Phi_2 \cup \{s\}\}.$$

现在设 $v \in \Phi_2 \cup \{s\}$ 和 $x \in \Phi \cap$ 袋$'(v)$。根据袋$'$的定义， x 要么：

- 等于 v。那么我们有 $x \in \Phi_2 \cup \{s\}$ 和 $x \in \Phi$，因此必然有 $x = s$；或者
- 是 v 的一个祖先，它通过一条边与 v的一个后代 y 相连。但由于还有 $x \in \Phi$，我们得出 x 也是 s的一个祖先。此外， y 是 s 的一个后代（因为 y 是 v 的后代，而 v是 s的后代）。我们得出 $x \in$ 袋$'(s)$。

在这两种情况下，我们都有 $x \in$ 袋$'(s)$，因此

$$\Phi \cap X \subseteq \mathtt{bag}'(s).$$

由于 $(F,$ 袋$')$ 是一个宽度为 $\text{tw}(G) \leqslant k \leqslant \ell$ 的分解，命题的结论立即成立。$\lhd$

命题 4.6。 设 t 是 T_{pref} 的一个附属节点。那么

$$|\mathtt{cmp}(t) \cap X| \leqslant (\ell+1) f(\ell).$$

命题证明。 根据去交替引理（引理 4.3）， $\mathtt{cmp}(t)$ 可以划分为至多$f(\ell)$ 个互不相交的因子，每个因子包含 F。根据 命题 4.5，每个这样的因子与 X的交集至多包含 $\ell+1$ 个元素。 $\lhd$

小闭包引理的证明可直接由断言 4.4 和4.6得出。 $\square$

4.2 最小权重闭包与闭包连通性引理

在确立了对于足够大的 c， c-小闭包的存在性之后，我们现在给出关于这类闭包的一个结构性结果：闭包连通性引理。直观地说，我们证明，若 $X \supseteq W$ 是一个 c-小闭包，且相对于某个度量而言是 W 最优的，那么 C 的每个连通分量$G - X$ 都与 W 良好连通。我们可以将此性质视为 Korhonen 和 Lokshtanov 的一项工作中类似结果的对应版本 [41, 引理 5。1]：区别在于，我们讨论的是最优的c-小闭包，而 [41]中讨论的只是最优闭包。

从现在起，令 $\omega\colon\ V(G) \to \mathbb{Z}$ 为任意权函数。对于任意子集 $A \subseteq V(G)$，令$\omega(A)$:$= \sum_{v\in A} \omega(v)$。首先，我们来定义几个涉及权函数的概念：

定义 5。固定 $k, c \in \mathbb{N}$和权函数 $\omega\colon\ V(G) \to \mathbb{Z}$。令 G 为一个树宽至多为 k, $W \subseteq V(G)$的图，并令 X 为 W的一个 c-小闭包。我们称 X 是 ω-极小的，若对于 W的每个c-小闭包 X'，以下条件之一成立：

- $|X'| > |X|$, *or*
- $|X'| = |X|$ *and* $\omega(X') \geqslant \omega(X)$.

定义 6([41])。令 G 为一个图， $A, B \subseteq V(G)$，且 $\omega\colon\ V(G) \to \mathbb{Z}$ 为一个权函数。若集合 A 满足以下条件之一，对任意 B 中的每个(A, B)-分隔集 S 均成立，则称该集合 ω-链接于 B：

- $|S| > |A|$,
- $|S| = |A|$ 和 $\omega(S) \geqslant \omega(A)$。

根据定义，如果 A 是 ω-链接到 B中的，那么 A 也链接到 B中。
现在我们可以陈述并证明闭包连通性引理：

引理 *4.7*（闭包连通性引理）。固定 k、 $c \in \mathbb{N}$ 和一个权函数 $\omega\colon\ V(G) \to \mathbb{Z}$。设$G$ 是一个树宽至多为 k的图，*(*T*,* 袋*)*是 G 的一个树分解（宽度任意）， T_{pref} 是 T的一个前缀，并且 X是袋*(*T_{pref}*)*的一个 ω-最小 c-小 k-闭包。则对于每个 $C \in cc(G - X)$，集合 $N(C)$ ω-链接到袋*(*T_{pref}*)*中。

我们注意到，由引理 4.7 和小闭包引理可知，如果分解 $(T$,袋$)$ 的宽度为 $\ell \geqslant k$，那么对于某个足够大的常数 $c_\ell \in \mathcal{O}(\ell^4)$ 存在一个 c_ℓ-小 k-闭包 X，其由 袋(T_{pref}) 组成，使得$G - X$ 的每个连通分量的邻域 ω-链接到 袋(T_{pref})。事实上，任何这样的 ω-最小闭包都具有这一性质。

闭包连通性引理的证明通过假设某个集合 $N(C)$ 未被ω-链接到袋(T_{pref}) 中来进行；此时，将存在一个小的 $(N(C)$，袋$(T_{\text{pref}}))$-分隔集。该分隔集将用于构造一个权重更小的袋(T_{pref}) 的新 k-闭包 X'，从而与 X 的 ω-极小性矛盾。然而，为了证明 X' 是袋(T_{pref}) 的 k-闭包，需要证明躯干图$_G(X')$ 的树宽足够小。为便于进行这一论证，我们将使用 Korhonen 和 Lokshtanov [41] 工作中的一个有用技术工具：

引理 *4.8*（拉取引理， [41, 引理 *4.8*]）。设 G 是一个图， $X \subseteq V(G)$ 和 *(*T*,* 袋*)* 是躯干图$_G(X)$ 的一个树分解。设 *(*A, S, B*)* 是 G 的一个分离，并满足以下条件：存在一个节点 $r \in V(T)$，使得 S 被链接到袋$(r) \cap (S \cup B)$ 中。另设 $X' := (X \cap A) \cup S$。那么，存在一个躯干图$_G(X')$ 的树分解 *(*T*,* 袋*′)*，其宽度不超过 *(*T*,* 袋*)* 的宽度。

我们还需要一个简单的辅助引理：

引理 4.9. 设 G 是一个图，$X \subseteq V(G)$ 且 $C \in \mathrm{cc}(G-X)$。则 $N(C)$ 是躯干图$_{G}(X)$中的一个团。

证明。对于每个 $u,\ v \in N(C)$，都存在一条连接 u 和 v 的路径，其所有内部顶点均包含在 C中。□

我们现在可以证明闭包连通性引理了。

引理 4.7 的证明。令 $W := \text{袋}(T_{\mathrm{pref}})$。为导出矛盾，假设存在一个分量 $C \in \mathrm{cc}(G - X)$，使得其邻域 $N(C)$无法 ω-链接到 W中。根据定义 6，存在一个 $(N(C)\ W)$-分隔集 S，使得要么 $|S| < |N(C)|$，要么 $|S| = |N(C)|$ 且$\omega(S) < \omega(N(C))$。不失一般性，假设 S 是规模尽可能小的此类分隔集。自然地，S诱导出一个分离 (A, S, B)，使得 $W \subseteq A \cup S$且 $N(C) \subseteq S \cup B$。

现在，按如下方式从 X 构造一个新的集合 X'：

$$X' := (X \cap A) \cup S.$$

我们声称，X' 也是 W的一个 c-小 k-闭包。注意，由于 $N(C)$包含于 X 且与 $X \cap A$不相交，因此有 $X' \subseteq (X \setminus N(C)) \cup S$。因此，$|X'| \leqslant |X| - |N(C)| + |S|$ 且$\omega(X') \leqslant \omega(X) - \omega(N(C)) + \omega(S)$。所以，如果 $|S| < |N(C)|$，则有 $|X'| < |X|$；如果 $|S| = |N(C)|$且 $\omega(S) < \omega(N(C))$，则 $|X'| \leqslant |X|$ 且 $\omega(X') < \omega(X)$。因此，只要 X' 确实是 W的一个 c-小k-闭包，X 就不可能是 ω-极小的，这与我们的假设矛盾。

断言 4.10. X' 是 W的 k-闭包。

命题证明. 由于 W 与 B 不相交，且 X 是 W的 k-闭包，因此我们得到 $X' \supseteq W$。

为了使用拉取引理，我们令$(T_X, \text{袋}_X)$为躯干图$_{G}(X)$的一个宽度至多为 $2k+1$的树分解。由引理 4.9，$N(C)$是躯干图$_{G}(X)$中的一个团，因此存在一个节点$r \in V(T_X)$，使得 $N(C) \subseteq \text{袋}_X(r)$。接下来只需验证 S被链接到袋$_X(r) \cap (S \cup B)$中。

注意到 $N(C) \subseteq \text{袋}_X(r) \cap (S \cup B)$。因此，只需检查 S是否被链接到 $N(C)$中。然而，如果情况并非如此，那么就存在一个规模为 $|S'| < |S|$的$(N(C)\ S)$-分隔集 S'。但这样一来，S'也会是一个规模小于 $|S|$的$(N(C)\ W)$-分隔集，这与 S的极小性矛盾。因此，引理 4.8适用于树分解$(T_X, \text{袋}_X)$和分离(A, S, B)，从而得到一个宽度不超过 $2k+1$的躯干图$_{G}(X')$的树分解。◁

断言 4.11. X' 是 c-小。

命题证明。回忆 $X \supseteq W$，其中 $W = \text{袋}(T_{\mathrm{pref}})$。因此，由于 C 是 $G - X$ 的一个连通分量，整个连通分量 C 必须包含在 T_{pref} 的某个附属节点 t 的 $\mathrm{cmp}(t)$ 中。因此，$N(C) \subseteq \mathrm{cmp}(t) \cup \mathrm{adh}(t)$。此外，$\mathrm{adh}(t) \subseteq W$。由于 $(\mathrm{cmp}(t), \mathrm{adh}(t)\ V(G) \setminus (\mathrm{cmp}(t) \cup \mathrm{adh}(t)))$ 是 G 的一个分离，且 S 是一个规模最小的 $(N(C)\ W)$-分隔集，因此可得 $S \subseteq \mathrm{cmp}(t) \cup \mathrm{adh}(t)$；换言之，$\mathrm{cmp}(t) \cup \mathrm{adh}(t)$ 之外的顶点对于将 $N(C)$ 与 W 分离并无助益。

现在，对于 T_{pref} 的一个附属节点 $t' \neq t$，$\mathrm{cmp}(t')$ 与 $\mathrm{cmp}(t) \cup \mathrm{adh}(t)$ 不相交：这是因为 $\mathrm{cmp}(t')$ 与 W（其中完整包含 $\mathrm{adh}(t)$）不相交，并且与 $\mathrm{cmp}(t)$ 也不相交（因为在 T 中，t、t' 不再保持祖先-后代关系）。因此，S 与 $\mathrm{cmp}(t')$ 不相交，从而根据 X' 的定义，

$$|X' \cap \mathtt{cmp}(t')| = |(X \cap A) \cap \mathtt{cmp}(t')| \leqslant |X \cap \mathtt{cmp}(t')| \leqslant c.$$

因此，附属节点 t'不违反小规模条件。

为了证明附属节点 t 满足相同的条件，我们注意到 $N(C) \cap W \subseteq S$（因为 S 将 $N(C)$ 与 W 分隔开来），因此 $N(C) \cap \mathsf{adh}(t) \subseteq S$。所以，$|N(C) \cap \mathsf{adh}(t)| \leqslant |S \cap \mathsf{adh}(t)|$，从而得到

$$|N(C) \cap \mathtt{cmp}(t)| = |N(C)| - |N(C) \cap \mathtt{adh}(t)| \geqslant |S| - |S \cap \mathtt{adh}(t)| = |S \cap \mathtt{cmp}(t)|.$$

现在，由于 $X' \subseteq (X \setminus N(C)) \cup S$ 和 $N(C) \subseteq X$，我们推断出

$$|X' \cap \mathtt{cmp}(t)| \leqslant |X \cap \mathtt{cmp}(t)| - |N(C) \cap \mathtt{cmp}(t)| + |S \cap \mathtt{cmp}(t)| \leqslant |X \cap \mathtt{cmp}(t)| \leqslant c. \lhd$$

根据 命题 4.10 和 4.11 我们推断出 X' 是一个 c-小 k-闭包 W。这与 ω-极小性相矛盾，X。 □

4.3 闭包探索、阻塞集和收集的连通分量

在本节中，我们定义若干辅助对象；在找到合适的闭包后，树分解改进算法将构造这些对象：探索集、阻塞集和收集的连通分量。在本节余下部分，固定一个宽度至多为 ℓ 的 G 的树分解（T，袋）、T 的一个非空前缀 T_{pref}，以及某个权函数 $\omega\colon \mathbb{N} \to \mathbb{N}$ 下的一个 ω-极小 c-小 k-闭包 $X \supseteq \mathsf{bags}(T_{\mathrm{pref}})$。

阻塞集。 我们从阻塞集的定义开始：

定义 7。 如果以下情况之一成立，我们称节点 $t \in V(T) \setminus T_{\mathrm{pref}}$ 是关于 T_{pref} 和 X 的 T 中的一个阻塞：

- 袋$(t) \subseteq N_G[C]$ 对于某个连通分量 $C \in \mathsf{cc}(G - X)$，该连通分量与袋(t) 相交（分量阻塞）；
- 袋$(t) \subseteq X$ 和袋(t) 是躯干图$_G(X)$ 中的一个团（团阻塞）；

且 t 的任何严格祖先都不是阻塞。

注意，一个分量阻塞恰好与一个连通分量 $C \in \mathsf{cc}(G - X)$ 相交。

由于上下文通常会给出 T、T_{pref} 和 X，我们通常只说 t 是一个阻塞。接下来引入以下记号：令 $\mathsf{Blockages}(T_{\mathrm{pref}}, X)$ 表示阻塞的集合。

阻塞的性质。 现在我们证明几个关于阻塞与 X之间关系的引理。

引理 4.12。 如果 $t \in$阻塞集(T_{pref}, X)，那么对于每一对顶点 $u, v \in$袋(t)，存在一条 uv-路径于 G 中，且与 X 内部不相交。

证明。 如果 t 是连通分量 $C \in \mathsf{cc}(G - X)$ 的分量阻塞，那么引理的结论可由 $\mathsf{bag}(t) \subseteq N_G[C]$ 立即推出：事实上，对于所有 u，$v \in \mathsf{bag}(t)$，存在一条 uv-路径，其所有内部顶点都属于 C。另一方面，如果 t 是团阻塞，那么该结论等价于断言 $\mathsf{bag}(t)$ 是躯干图$_G(X)$ 中的一个团。 □

注意到，由引理 4.12 可立即推出：如果 t 是一个阻塞集，那么袋$(t) \cap X$ 是躯干图$_G(X)$中的一个团。

引理 4.13。如果 $t \in$ 阻塞集$(T_{\text{pref}},\ X)$，那么$\texttt{cmp}(t) \cap X = \emptyset$。

证明。反设不然。于是我们声称，集合 $X' := X \setminus \text{cmp}(t)$ 也是 bags(T_{pref}) 的一个 c-小 k-闭包，这与 X 的极小性矛盾。事实上，我们将证明 torso$_G(X')$ 是 torso$_G(X)$ 的子图。为此，任选顶点 $u, v \in X'$，并假设 $uv \in E(\text{torso}_G(X'))$，也就是说，存在一条路径 $P = v_1, v_2, \ldots, v_p$，从 $u = v_1$ 到 $v = v_p$，且与 X' 内部不相交。若 P 与 X 内部不相交，则也有 $uv \in E(\text{torso}_G(X))$，证明完毕。反之，$P$ 必与 $X \setminus X'$ 相交，因此也与 cmp(t) 相交。注意，根据 X' 的定义，必有 $u, v \notin \text{cmp}(t)$。令 v_a, v_b $(1 \leqslant a \leqslant b \leqslant p)$ 分别为 P 与 adh(t) 的第一个和最后一个交点；这些顶点确实存在，因为 (cmp(t), adh(t), $(V(G) \setminus (\text{cmp}(t) \cup \text{adh}(t)))$ 是 G 的一个分离。子路径 $v_1, v_2, \ldots, v_a$ 和 $v_b, v_{b+1}, \ldots, v_p$ 与 cmp(t) 不相交，因此与 X 内部不相交。通过拼接以下部分，从 u 到 v 构造一条新路径 P'：

- 子路径 $v_1,\ v_2,\ \ldots\ldots,\ v_a$,
- 一条从 v_a 到 v_b 与……内部不相交的 X (其存在性由引理 4.12断言),
- 子路径 $v_b,\ v_{b+1},\ \ldots\ldots,\ v_p$。

显然，P' 再次与 X'内部不相交。我们断言，P' 与 X内部不相交，从而证明 $uv \in E$(躯干图$_G(X)$)。事实上，P' 的三个线段都与 X内部不相交，因此，只有当 $v_a \in X$ (或 $v_b \in X$) 时，P' 才可能与 X 内部相交。然而，在这种情况下，我们有 $v_a \in X'$ （分别为 $v_b \in X'$），因为 $v_a,\ v_b \notin$ cmp(t)。但是，P' 与X'内部不相交，因此 v_a （分别为 v_b）必须是 P' 的第一个（分别为最后一个）顶点，因而不是 P'的内部顶点。因此，P' 与 X内部不相交。 □

我们注意到，如果 t 是一个团阻塞，那么由引理 4.13 可知袋$(t) = \texttt{adh}(t)$，因此特别有袋$(t) \subseteq$ 袋(父节点(t))。

探索与探索图。对于给定的权函数 ω: $V(G) \to \mathbb{Z}$、T 的前缀 T_{pref}以及由袋构成的 ω-极小 c-小 k- 闭包 X (T_{pref})，我们将探索定义为 T 的前缀 Exploration(T_{pref}, X)，其附属子树集合由 Blockages(T_{pref}, X)给出。若节点 $t \in V(T)$ 满足 $t \in$ Exploration(T_{pref}, X)，则称其已探索，否则称其未探索。接着，若顶点$v \in V(G)$属于某个已探索节点 t的袋 bag(t)，则称其已探索，否则称其未探索。注意，当且仅当 $v \in V(G)$属于某个阻塞 t的 cmp(t) 时，它未被探索。特别地，根据引理 4.13，X 中的每个顶点都已探索。

显然，我们有 $T_{\text{pref}} \subseteq$ 探索(T_{pref}, X)。此外，若 T是一个二叉树分解，则|阻塞集$(T_{\text{pref}}, X)| \leqslant |$探索$(T_{\text{pref}}, X)| + 1$(这由 事实 3.1 立即得出)。

接下来，我们通过将阻塞 t 的连通分量 cmp(t) 压缩为单个顶点，来定义探索图 $H :=$ ExplorationGraph(T_{pref}, X)。此定义旨在以一种其连通分量数量可由 |Exploration(T_{pref}, X)| 界定的方式呈现 $G - X$ 的连通分量，即使这些连通分量的数量可能大得多。形式化地说：

- $V(H)$ 包含：已探索顶点，即顶点集 $\bigcup_{t \in \text{探索}(T_{\text{pref}} X)}$袋$(t)$；以及阻塞顶点，即图 T 的节点集合 Blockages$(T_{\text{pref}}\, X)$。
- 边集合 $E(H)$ 的构造方式是：取由已探索顶点诱导的 G 子图，并为每个 $t \in$ 阻塞集$(T_{\text{pref}}\, X)$ 和 $v \in$ adh(t) 添加边 tv。

• 压缩映射 ξ: $V(G) \to V(H)$ 在 $V(G) \cap V(H)$ 上是恒等映射；对于 $t \in$ 阻塞集$(T_{\text{pref}}\, X)$, 我们设置 $\xi^{-1}(t) \coloneqq \mathsf{cmp}(t)$。

注意， t 引理 4.13 蕴含 $X \subseteq V(H)$。此外，若 T是一个二叉树分解， n

$$|V(H)| \leqslant (\ell+1) \cdot |\mathsf{Exploration}(T_{\text{pref}}, X)| + |\mathsf{Exploration}(T_{\text{pref}}, X)| + 1.$$

我们还观察到以下事实：

引理 4.14。 若 $uv \in E(G)$，则 $\xi(u) = \xi(v)$ 或 $\xi(u)\xi(v) \in E(H)$。

证明。若 u 和 v 均已探索，则 $\xi(u) = u$， $\xi(v) = v$ 且 $\xi(u)\xi(v) \in E(H)$。否则，不失一般性，假设 $\xi(v) = t$ 是一个阻塞顶点。那么 $v \in \mathsf{cmp}(t)$，因此 $u \in \mathsf{cmp}(t) \cup \mathsf{adh}(t)$。若 $u \in \mathsf{cmp}(t)$，则 $\xi(u) = \xi(v) = t$。否则 $u \in \mathsf{adh}(t)$，因此 u 是一个已探索顶点。于是 $\xi(u) = u$ 且 $\xi(v) = t$，所以根据构造， $\xi(u)\xi(v) \in E(H)$。 □

收集的连通分量。 使用上述记号，我们将 $V(G) \setminus X$ 划分为称为收集的连通分量 的子集，具体如下。对于每个 $C \in \mathsf{cc}(H - X)$，令 $\mathfrak{C} \coloneqq \xi^{-1}(C)$（因此 $\mathfrak{C} \subseteq V(G) \setminus X$），若$\mathfrak{C}$ 非空，则将 $\mathfrak{C}$ 纳入 G的收集连通分量集合 $\mathsf{Coll}(T_{\text{pref}}, X)$。换言之，我们对 $H - X$ 的每个连通分量进行解压，形成一个收集分量——$V(G) \setminus X$的一个子集。容易验证，集合 $\mathsf{Coll}(T_{\text{pref}}, X)$ 确实是 $V(G) \setminus X$的一个划分。

设 $\mathfrak{C} \in \mathsf{Coll}(T_{\text{pref}}, X)$。我们称 $\mathfrak{C}$ 为：

• 未阻塞 ， 如果 $\xi(\mathfrak{C})$ 至少包含 $V(H)$ 的一个已探索顶点；或者

• 受阻 ， 如果 $\xi(\mathfrak{C})$ 仅包含 $V(H)$ 的阻塞顶点。

注意，如果 t 是团阻塞，那么 t 是 $H - X$的一个孤立顶点；因此，收集分量 $\xi^{-1}(\{t\})$（如果非空）必然是一个受阻分量。然而，反向推论一般不成立：可能存在一个分量阻塞 t' ，满足 $\mathsf{adh}(t') \subseteq X$，从而再次导致 t' 成为 $H - X$的一个孤立顶点。此时，收集分量 $\xi^{-1}(\{t'\})$（如果非空）也将是一个受阻分量。

事实证明， $\mathsf{Coll}(T_{\text{pref}}, X)$ 是 $V(G) \setminus X$ 比 $\mathsf{cc}(G - X)$ 更粗的划分：

引理 4.15. 每个收集分量 $\mathfrak{C} \in \mathsf{Coll}(T_{\text{pref}}, X)$ 都是 $G - X$ 的一组连通分量的并集。

证明。 只需证明以下结论：若 $u, v \in V(G) \setminus X$ 在 G中相邻，那么 $\xi(u)$ 和 $\xi(v)$属于 H的同一个连通分量。但根据 引理 4.14， $\xi(u)$ 和 $\xi(v)$ 在 H中相等或相邻；这就证明了该结论。 □

接下来，我们证明每种类型的收集分量的刻画。

引理 4.16. 若 $\mathfrak{C} \in \mathsf{Coll}(T_{\text{pref}}, X)$ 满足：

• 未阻塞，则属于 $\xi(\mathfrak{C})$ 的已探索顶点集合包含于单个连通分量 $C \in \mathrm{cc}(G - X)$中。此外，$C \subseteq \mathfrak{C}$ 和 $N(\mathfrak{C}) = N(C)$；

• 受阻，则 ξ（$\mathfrak{C}$）由单个阻塞顶点 t 构成。此外， N（$\mathfrak{C}$） $\subseteq$ adh（t） $\cap X$。

证明。我们先证明以下命题：

命题 4.17。若 u、 $v \in \mathfrak{C}$ 已被探索，因此 u 和 v 属于 $G - X$的同一个连通分量。

命题证明。 设 $P = w_1 w_2 \ldots w_p (w_1 = \xi(u),\ w_p = \xi(v))$是 $H - X$ 中从 $\xi(u)$到$\xi(v)$的一条简单路径（由于 u、 v 位于同一个收集分量中，这样的路径存在）。我们将归纳证明， P的每个已探索顶点都与 w_1位于 $G-X$的同一个连通分量中。对于 w_1，这显然成立。归纳步骤中，选取一个已探索顶点 w_i $(i \geqslant 2)$。若 w_{i-1} 也已被探索，则 $w_{i-1}w_i$ 是 $G - X$的一条边，因此 w_i 与w_{i-1} 位于 $G - X$ 的同一个连通分量中（因而也与 w_1位于 $G - X$ 的同一个连通分量中）。

另一方面，若 w_{i-1} 是阻塞顶点（即 $w_{i-1} \in V(T)$），则注意到 $i \geqslant 3$（因为 w_1是已探索顶点）。此外，根据 $E(H)$ 的定义， w_{i-1} 在 H 中仅与 $\mathsf{adh}(w_{i-1})$ 内的已探索顶点相邻。因此， w_{i-2} 也是已探索顶点，且 $w_{i-2},\ w_i \in \mathsf{adh}(w_{i-1})$。由于$w_{i-2},\ w_i \notin X$，可知 w_{i-1} 是分量阻塞，即 $\mathsf{bag}(w_{i-1}) \subseteq N_G[C]$，其中某个连通分量 $C \in \mathsf{cc}(G - X)$ 与 $\mathsf{bag}(t)$ 相交。因此， w_{i-2} 和 w_i 都属于 C；根据归纳假设， w_1也属于其中。◁

此外，下面这个简单事实也很有用：

命题 4.18。 若 D 是 $G - X$ 的一个连通分量，且对于某个阻塞 t，有 $D \subseteq \mathtt{cmp}(t)$，则$N(D) \subseteq \mathtt{adh}(t) \cap X$。

命题证明。 显然， $N(D) \subseteq X$。然而，根据引理 4.13，$\mathtt{cmp}(t) \cap X = \emptyset$。由于 $N[D] \subseteq \mathtt{cmp}(t) \cup \mathtt{adh}(t)$，因此 $N(D) \subseteq \mathtt{adh}(t)$。◁

现在，假设 $\mathfrak{C}$ 未阻塞。根据命题 4.17，属于 $\xi(\mathfrak{C})$ 的已探索顶点包含在单个连通分量 $C \in \mathrm{cc}(G - X)$中。由引理 4.15，我们有$C \subseteq \mathfrak{C}$。还需证明 $N(\mathfrak{C}) = N(C)$。为此，我们将论证以下结论：

命题 4.19。 令 $D \in \mathrm{cc}(G - X)$ 为一个满足 $D \subseteq \mathfrak{C}$的连通分量。则 $N(D) \subseteq N(C)$。

命题证明。 我们可以假设 $D \neq C$。那么集合 $\xi(D)$ 仅包含阻塞顶点；由于阻塞顶点在 H中构成一个独立集，因此根据引理 4.14 可知 $\xi(D)$ 只包含一个阻塞顶点，记为 t。因此 $D \subseteq \mathtt{cmp}(t)$，并且根据命题 4.18还可知 $N(D) \subseteq \mathtt{adh}(t) \cap X$。

令 $u \in C$ 为任意一个已探索顶点（因此 $\xi(u) = u$）。现在，由于 $C,\ D \subseteq \mathfrak{C}$ 和 $\xi(D) = \{t\}$，必然存在一条路径 P，从 t 通向 u，且位于 $H - X$ 中。令 v 为 P 中紧接在 t 之后的（已探索）顶点。根据命题 4.17，$v \in C$；并且根据构造， $v \in \mathsf{adh}(t)$。由此可知， t 是一个与 $C \in \mathsf{cc}(G - X)$ 相交的分量阻塞，因此 $\mathsf{bag}(t) \subseteq N[C]$；特别地，$\mathsf{adh}(t) \cap X \subseteq N(C)$。◁

作为 $N(\mathfrak{C}) = \bigcup_{D \in \mathrm{cc}(G-X),\, D \subseteq \mathfrak{C}} N(D)$，由 命题 4.19 可得 $N(\mathfrak{C}) = N(C)$，正如所需。

最后，假设 $\mathfrak{C}$ 受阻。由于 H 的阻塞顶点构成一个独立集，根据引理 4.14 可知，$|\xi(\mathfrak{C})| = 1$。令 t 为 $\xi(\mathfrak{C})$ 中唯一的元素，即该阻塞顶点。于是，命题 4.18 适用于每个带有 $D \subseteq \mathfrak{C}$ 的 $D \in \mathsf{cc}(G - X)$，并推出 $N(D) \subseteq \mathsf{adh}(t) \cap X$。因此， $N(\mathfrak{C}) \subseteq \mathsf{adh}(t) \cap X$。□

下一个引理可由前一个引理轻易推出：

引理 4.20. 对于每个 $\mathfrak{C} \in \mathsf{Coll}(T_{\mathrm{pref}}, X)$，集合 $N(\mathfrak{C})$ 是躯干图$_G(X)$中的一个团。

证明。如果 $\mathfrak{C}$ 未阻塞，那么对于某个 $C \in \mathsf{cc}(G - X)$， $N(\mathfrak{C}) = N(C)$。根据引理 4.9， $N(C)$ 是躯干图$_G(X)$中的一个团。另一方面，如果 $\mathfrak{C}$ 受阻，那么对于某个阻塞 t， $N(\mathfrak{C}) \subseteq \mathsf{adh}(t) \cap X$。但根据阻塞的定义可立即得到，$\mathsf{adh}(t) \cap X$ 是躯干图$_G(X)$中的一个团。□

最后，我们将收集分量 $\mathfrak{C}$ 的归属袋 定义为按如下方式选取的 T 的一个节点：

- 若 $\mathfrak{C}$未受阻，则 $\mathfrak{C}$的归属袋是 T中满足以下条件的最浅节点 t：bag(t) 包含 $\mathfrak{C}$中的一个已探索顶点。注意，该袋是唯一确定的，因为 $\mathfrak{C}$中的已探索顶点诱导出 G 的一个连通子图，因此它们在 (T,袋) 中的出现位置诱导出 T的一棵连通子树。

- I若 $\mathfrak{C}$ 受阻，则 $\mathfrak{C}$ 的归属袋是唯一满足 $\xi(\mathfrak{C}) = \{t\}$的阻塞 t .

最后，我们证明收集分量的归属袋的一些性质 s.

L引理 4.21。 设 $\mathfrak{C} \in \mathsf{Coll}(T_{\mathrm{pref}}, X)$ 未受阻，且 t 是 $\mathfrak{C}$的归属袋。则

$$t \in \mathsf{Exploration}(T_{\mathrm{pref}}, X) \setminus T_{\mathrm{pref}} \qquad \textit{and} \qquad |\mathsf{bag}(t)| > |N(\mathfrak{C})|.$$

证明。 首先注意， t 是一个已探索节点： $\mathfrak{C}$ 的每个已探索顶点都包含在某个已探索节点 u 对应的袋 bag(u) 中，而已探索节点集合是祖先闭合的。此外， $t \notin T_{\mathrm{pref}}$ 显然成立：根据定义，我们有 bags(T_{pref}) $\subseteq X$ ，但 $\mathfrak{C}$ 与 X不相交。由此可得$t \in$ Exploration(T_{pref}, X) $\setminus T_{\mathrm{pref}}$。

接下来我们证明 |bag(t)| $> |N(\mathfrak{C})|$。回顾引理 4.16，可知存在某个 $D \in$ cc($G - X$)，它包含 $\mathfrak{C}$ 的所有已探索顶点，并且满足 $N(\mathfrak{C}) = N(D)$。由于 t 是 T 的一个已探索节点，bag(t) $\setminus X$ 仅由已探索顶点组成。因此， t 是与 D 相交的 T 中最浅的袋。由于 $G[D]$ 连通，与 D 相交的 T 中所有袋都必须位于以 t 为根的子树 T 中。此外，根据闭包连通性引理（引理 4.7），$N(D)$ 与 bags(T_{pref}) 之间是 ω-链接的。由此，我们得到 adh(t) 是一个 ($N(D)$, bags(T_{pref}))-分隔集，因此可推出 |adh(t)| $\geqslant |N(D)|$。然而，adh(t) 不可能是 bag(t)，因为 adh(t) $\cap D = \emptyset$，而 bag(t) $\cap D \neq \emptyset$。因此，adh(t) $\subsetneq$ bag(t)，所以 |bag(t)| $> |N(D)|$。 □

引理 4.22。 设 $\mathfrak{C} \in \mathsf{Coll}(T_{\mathrm{pref}}, X)$ 受阻，且 t 是 $\mathfrak{C}$ 的归属袋。假设父节点$(t) \notin T_{\mathrm{pref}}$。那么 |袋 $(t) \cap X|$ < |袋 (父节点 (t))|。

证明。由于 $\mathfrak{C}$ 受阻， t 是一个阻塞。由引理 4.13 可知 cmp(t)$\cap X = \emptyset$，因此 bag(t)$\cap X$ =adh(t)$\cap X$。但 adh(t)$\cap X$ = bag(parent(t)) 不可能成立：否则，由于 adh(t)$\cap X$ 是躯干图$_G(X)$中的一个团（引理 4.12），parent(t) 就会是一个团阻塞，而这样一来 t 本身就不可能是一个阻塞。由于 adh(t) $\subseteq$ bag(parent(t))，我们得出 |bag(t)$\cap X$| < |bag(parent(t))|。 □

4.4 计算闭包和辅助对象

最后，我们说明，在因前缀重建更新而动态变化的树分解中，可以高效地计算前几节定义的对象。回顾一下，这类分解由带注释的树分解建模，而用于维护这类动态分解的数据结构称为前缀重建数据结构。本节中，我们固定一个特定的权函数：对于图 G 的带注释树分解 $\mathcal{T}$ =(T，袋，边)，我们采用 $\mathcal{T}$ 的深度函数 $d_{\mathcal{T}}$作为权函数。

现在我们可以陈述此前承诺的结果了。

引理 4.23。对于每个 c、ℓ、$k \in \mathbb{N}$，满足 $\ell \geqslant k$，都存在一个 ℓ-前缀重建数据结构，其开销为 $2^{\mathcal{O}((c+\ell)^2)}$ ，并且还支持以下操作：

- 查询($T_{\text{前缀}}$)：给定 $T_{\text{前缀}}$ T，若不存在 袋($T_{\text{前缀}}$)的 c-小 k-闭包，则返回 *No closure*；否则

– 一个 $d_{\mathcal{T}}$-极小的 c-小闭包 X 袋(T_{pref})，– 图躯干图$_G(X)$，以及– 集合 阻塞集$(T_{\text{pref}}, X) \subseteq V(T)$。

该操作的最坏情况时间为 $2^{\mathcal{O}(k(c+\ell)^2)} \cdot (|T_{\text{pref}}| + |$探索$(T_{\text{pref}}, X)|)$。

引理 4.23 的证明见附录 A.4。

5 细化数据结构

本节将定义一种称为细化数据结构的数据结构，它将在第 6 节 和7 中用于改进给定动态图的树分解。

设 G 是一张树宽至多为 k 的图，它随时间推移通过边插入和删除而变化。本工作的最终目标是维护 G 的一个宽度至多为 $6k+5$ 的带注释的二叉树分解$\mathcal{T} = (T,$ 袋, 边)（当不使用边函数时，我们通常写作 $\mathcal{T} = (T,$ 袋)）。细化操作有两个目的。首先，它用于改进分解的宽度；也就是说，给定 $(T,$袋) 的一个前缀 T_{pref}，其中包含所有规模大于 $6k+6$ 的袋，细化操作会输出一个宽度至多为 $6k+5$ 的树分解 $(T',$ 袋′)。其次，当分解的树变得过于不平衡时，也会调用该操作。我们将在第 6 节中对此作进一步说明。

在继续介绍细化数据结构之前，我们先引入势函数。它在分析细化操作的摊还复杂度以及第 6 节的高度缩减方案中都发挥着重要作用。

5.1 势函数

设 $k \in \mathbb{N}$ 为动态图树宽的上界，$\ell = 6k+5$为我们所维护的树分解所期望的宽度（特别是，调用细化操作时，实际宽度至多为 $6k+6$ ）。对于树分解$\mathcal{T} = (T,$ 袋)中的节点 t，我们用以下公式定义其势函数：

$$\Phi_{\ell,\mathcal{T}}(t) := g_\ell(|\mathsf{bag}(t)|) \cdot \mathsf{height}_T(t), \tag{1}$$

其中

$$g_\ell(x) := (53(\ell+4))^x \qquad \text{for every } x \in \mathbb{N}.$$

注意，由于我们维护的分解宽度为 $\mathcal{O}(k)$，因此有

$$\Phi_{\ell,\mathcal{T}}(t) \leqslant 2^{\mathcal{O}(k \log k)} \cdot \mathsf{height}_T(t) \qquad \text{for every node } t.$$

直观而言，势函数中的这一项 $g_\ell(|$袋$(t)|)$ 使我们能够通过用 $\mathcal{O}(\ell)$ 个 t副本替换节点 t，其中每个副本 t' 的高度都与 t 相同，但 t' 的袋严格小于 t的袋，从而更新树分解。这样，我们就可以用势函数的减少来支付此类变换的代价。第二项 高度$_T(t)$ 将在 第 6 节中发挥关键作用：如果当前树分解过于不平衡，它将使我们能够论证势函数可以显著降低。

对于子集 $W \subseteq V(T)$，我们记作

$$\Phi_{\ell,\mathcal{T}}(W) := \sum_{t \in W} \Phi_{\ell,\mathcal{T}}(t),$$

类似地，对于整个树分解 $\mathcal{T} = (T, \text{袋})$，我们定义

$$\Phi_\ell(\mathcal{T}) \coloneqq \sum_{t \in V(T)} \Phi_{\ell,\mathcal{T}}(t).$$

5.2 数据结构

下一个引理描述了我们数据结构的接口。

引理 5.1 (细化数据结构)。 固定 $k \in \mathbb{N}$ 并令 $\ell = 6k+5$。存在一种开销为 $2^{\mathcal{O}(k^8)}$ 的 $(\ell+1)$ 前缀重建数据结构，它维护一个树分解 $\mathcal{T} = (T, \text{袋})$，其中有 $N \coloneqq |V(T)|$ 个节点，并且还支持以下操作：

- 细化$(T_{\text{前缀}})$：给定 T 的一个前缀 $T_{\text{前缀}}$，使得 $T_{\text{前缀}}$ 包含 $\mathcal{T}$ 的所有节点，且袋的规模为 $\ell+2$，返回一个前缀重建更新的描述 u，使得应用 u 后得到的树分解 $\mathcal{T}'$ 具有以下性质：

(WIDTH) $\mathcal{T}'$ 的宽度至多为 ℓ，并且

(POT) 以下不等式成立：

$$\Phi_\ell(\mathcal{T}) - \Phi_\ell(\mathcal{T}') \geqslant \sum_{t \in T_{\text{pref}}} \mathsf{height}_T(t) - 2^{\mathcal{O}(k \log k)} \cdot \left(|T_{\text{pref}}| + \sum_{t \in \mathtt{App}(T_{\text{pref}})} \mathsf{height}_T(t) \right) \cdot \log N.$$

此外，还有

(RT1) $\mathsf{refine}(T_{\text{pref}})$的最坏情况运行时间，因此 $|u|$ 的最坏情况运行时间也有上界

$$2^{\mathcal{O}(k^9)} \cdot \left(\left(|T_{\text{pref}}| + \sum_{t \in \mathtt{App}(T_{\text{pref}})} \mathsf{height}_T(t) \right) \cdot \log N + \max(\Phi_\ell(\mathcal{T}) - \Phi_\ell(\mathcal{T}'), 0) \right).$$

在本节余下部分，我们将介绍引理 5.1 中的数据结构，并证明后续将用到的一些关键性质。本节最后将证明细化操作的正确性，其中包括性质 (WIDTH)的证明。该数据结构的摊还分析，尤其是性质 (RT1) 和 (POT) 的证明，将在第 5.3 节中给出。

固定正整数 k, ℓ，使得 $\ell = 6k+5$。设 G 是一个动态的、具有 n 个顶点且树宽至多为 k 的图。在我们的数据结构中存储：

- 一个宽度至多为 $\ell+1$ 的 G 的二叉带注释树分解 $\mathcal{T} = (T, \text{袋}, \text{边})$；
- 一个用于 $\mathbb{D}_{\text{aux}}$ 的数据结构实例，来自引理 3.5，用于维护关于 $\mathcal{T}$ 的各种辅助信息；以及
- 一个实例 $\mathbb{D}$，用于探究引理 4.23 中的数据结构，以计算细化操作所需的必要对象。

对我们数据结构中的更新操作进行实现非常简单。收到带有前缀 T_{pref} 的前缀重建更新 u 后，我们根据 u 重新计算分解（T、袋、边），并将 u 传递给内部数据结构 $\mathbb{D}^{\text{aux}}$ 和 $\mathbb{D}^{\text{explore}}$。数据结构的初始化同样简单。

从现在起，我们关注细化操作。令 $T_{\text{pref}} \subseteq V(T)$ 为给定的 T前缀，其中包含 $\mathcal{T}$中规模为 $\ell + 2$的所有袋。

为便于说明，我们不构造 u 前缀重建更新的描述，而是展示如何在应用 $(T'$、袋$')$ 后，从 $(T$、袋$)$ 得到一个树分解 u。随后，可以从 u 以及计算过程中得到的对象中轻松提取出相应的描述；$(T'$、袋$')$ 特别是，这里将隐式地使用引理 3.7 。

细化操作分五个步骤进行。

步骤 1 (计算辅助对象)
给定分解 T_{pref} 的一个前缀 $\mathcal{T} = (T,$ 袋$)$ 我们使用 $\mathbb{D}^{\text{explore}}$ （出自引理 4.23 ）中的数据结构来计算以下对象：

- 一个 $d_{\mathcal{T}}$-极小 c-小 k-闭包 X，对应于 袋$(T_{\text{前缀}})$，其中 $d_{\mathcal{T}}$是 $\mathcal{T}$在 第 3 节 中定义的深度函数，而 $c \in \mathcal{O}(k^4)$是由引理 4.2给出的界，
- 图$_G(X)$的躯干图，以及
- 集合 Blockages$(T_{\text{前缀}}\, X) \subseteq V(T)$。

T然后，我们立即应用下面给出的引理 5.2来计算所有 o

- 已探索的前缀 F:= 探索$(T_{\text{前缀}}\, X)$，以及
- 探索图 H:= 探索图$(T_{\text{前缀}}\, X)$。

引理 5.2。 设 T_{pref} 是 T 的一个前缀，且 X 是 袋(T_{pref})的一个 $d_{\mathcal{T}}$-最小 c-小 k-闭包。给定 T_{pref}、 X和 阻塞集$(T_{\text{pref}}$、 $X)$，可以在时间 $k^{\mathcal{O}(1)} \cdot |F|$内计算出已探索的前缀 F和探索图 H。

证明。 首先，只需对 T 进行深度优先搜索并在遇到阻塞时停止，即可在 $\mathcal{O}(|F|)$ 时间内计算出 F。然后，通过取袋(F)与阻塞集$(T_{\text{pref}},\ X)$的并集，可以在 $k^{\mathcal{O}(1)} \cdot |F|$ 时间内计算出探索图的顶点 $V(H)$。已知 $(T,$ 袋，边$)$后，诱导子图 $G[$袋$(F)]$ 可在 $k^{\mathcal{O}(1)} \cdot |F|$ 时间内计算，因为它的所有边都存储在边(F)中。此外，阻塞顶点与袋(F)中的顶点之间的边可以直接根据定义计算。 □

显然，这一步的运行时间主要由对 $\mathbb{D}^{\text{explore}}$ 的调用决定；将 $c = \mathcal{O}(k^4)$ 代入查询的运行时间界中，可得如下结论。

(RT2) 步骤 1 的运行时间为 $2^{\mathcal{O}(k^9)} \cdot |F|$。

在进一步分析细化数据结构时，以下定义将很有用。

定义 8. 对于收集分量 $\mathfrak{C} \in$ Coll(T_{pref}, X)，我们将其接口、权重和高度分别记为Interface$(\mathfrak{C})$、weight$(\mathfrak{C})$ 和height$(\mathfrak{C})$，定义如下。

- 如果 $\mathfrak{C}$ 未阻塞，则集合接口$(\mathfrak{C})$: $= N(\mathfrak{C})$。

- 如果 $\mathfrak{C}$ 受阻，则集合接口($\mathfrak{C}$)：$=$袋(t) $\cap X$，其中 $t \in$阻塞集($T_{\mathrm{pref}} X$) 是 $\xi(\mathfrak{C})$的唯一元素（即 t 是 $\mathfrak{C}$的归属袋）。

在这两种情况下，我们设

$$\mathsf{weight}(\mathfrak{C}) := |\mathsf{Interface}(\mathfrak{C})| \qquad \textit{and} \qquad \mathsf{height}(\mathfrak{C}) := \mathsf{height}(t),$$

其中 t 是 $\mathfrak{C}$ 的归属袋。

此外，我们引入如下记号，将每个集合 $B \subseteq V$（G）映射到接口为 B 的收集分量集合：

$$\mathsf{Interface}^{-1}(B) := \{\mathfrak{C} \in \mathsf{Coll}(T_{\mathrm{pref}}, X) : \mathsf{Interface}(\mathfrak{C}) = B\}.$$

我们指出，这由 引理 4.16 可知： $N(\mathfrak{C}) \subseteq$接口($\mathfrak{C}$)。

步骤 ② (构建新前缀)
在这一步，我们打算构造新分解 $\mathcal{T}' = (T', 袋')$ 的一个前缀。结果表明，该前缀将是躯干图 $_G(X)$ 的一个树分解 $\mathcal{T}^X$，并具有一些额外性质。这里使用躯干图 $_G(X)$ 的原因很简单：对于 $G - X$ 的每个连通分量 C，邻域 $N(C) \subseteq X$ 都是躯干图 $_G(X)$ 中的一个团。因此，根据事实 3.4，对于躯干图 $_G(X)$ 的每个树分解 $(T^X, 袋^X)$，都存在一个节点 $t \in T^X$，其袋 bag(t) 包含 $N(C)$。这将使我们能够将躯干图 $_G(X)$ 的树分解与 $G - X$ 各连通分量的树分解结合起来。

下面详细描述所构造的前缀 $\mathcal{T}^X$。首先陈述以下两个已知结果，它们分别由 Bodlaender 以及 Bodlaender 和 Hagerup 提出。

定理 5.3 ([13, 定理 1]). 给定一个图 G，其中 $k := \mathrm{tw}(G)$，可以在时间 $_2^{\mathcal{O}(k^3)} \cdot |V(G)|$ 内计算出 G 的树分解 $(T, 袋)$，其宽度为 k，并且满足 $|V(T)| = \mathcal{O}(|V(G)|)$。

定理 *5.4*（[15, 引理 *2.2*]）。给定一个图 G 及其宽度为 k' 的树分解（T，袋），可以在时间 $\mathcal{O}(k' \cdot |V(T)|)$ 内计算出 G 的二叉树分解（T'，袋′），其高度为 $\mathcal{O}(\log |V(T)|)$，宽度至多为 $3k' + 2$，并且满足 $|V(T')| = \mathcal{O}(|V(T)|)$。

此外，我们将使用另一种辅助操作来修改二叉树分解。

引理 *5.5.* 给定一个图 G 及其高度为 h、宽度为 k'的二叉树分解 $(T, 袋)$，可以在 $\mathcal{O}(2^{k'} \cdot |V(T)|)$ 时间内计算出一个图 G 的二叉树分解 $(T', 袋')$，其高度为 $\mathcal{O}(h + k')$、宽度为 k'，且具有 $|V(T')| = \mathcal{O}(2^{k'} \cdot |V(T)|)$；使得对于每个节点 $t \in V(T)$ 和每个子集$B \subseteq$袋(t)，都存在 T' 的一个叶节点 t_B，满足 袋′(t_B) $= B$。

证明。令我们遍历所有节点 $t \in V(T)$。令袋(t) $= \{v_1, v_2, \ldots, v_p\}$，其中 $p \leqslant k'+1$。

我们按如下方式构造诱导子图 bag(t) 的二叉树分解 (T_t, bag_t)。T_t 的根是一个新节点 p，其袋为 $_t(p)$ $=$ bag(t)。T_t 的高度为 p。深度为 i 的每个顶点 $u \in V(T_t)$（其中 $i = 0, 1, \ldots, p-1$）都有两个子节点 u_{yes} 和 u_{no}，其袋分别为 $_t(u_{\mathrm{yes}})$ $=$ bag$_t(u)$ 和 $_t(u_{\mathrm{no}})$ $=$ bag$_t(u)$ $\setminus \{v_{i+1}\}$。这些性质定义了整个分解 (T_t, bag_t)。这是 bag(t) 的一个有效树分解，因为每当我们从 $u \in V(T_t)$ 的一个袋中排除顶点 v 时，v 就不会出现在 T_t 中 u 的任何后代的袋中。此外，请注意，对于每个子集 $B \subseteq$ bag(t)，都存在一个叶节点 $t_B \in V(T)$，使得 bag$_t(t_B)$ $= B$。

接下来，只需将这样的树 T_t 连接到原树 T 上——这将构成所需的分解（T'，袋′），其中袋′$|_T =$袋，且对于每个 t，袋′$|_{T_t} =$袋$_t$。如果一个顶点 t 至多有一个子节点，

那么我们只需将 T_t 的根设为 t的子节点，即可将 T_t 连接到 T。如果 t 有两个子节点，那么可以在它通向其中一个子节点的边上插入顶点 t_{copy}，使得袋′（t_{copy}）＝袋（t），然后对 t_{copy}应用前一种情况。

容易看出，(T'，袋′) 的宽度仍为 k'，而 T' 的高度至多为 $\mathcal{O}(h+k')$，因为每棵附加树 T_t 的高度至多为 $k'+1$。 □

现在可以定义所需的分解 $\mathcal{T}^X$，即躯干图$_G(X)$ 的分解。我们执行以下操作。

1. 应用 定理 5.3，得到躯干图$_G(X)$ 的一个树分解 $\mathcal{T}^X{}_{\text{opt}}$。
2. 对分解 $\mathcal{T}^X{}_{\text{opt}}$ 使用定理 5.4 中的算法，并将所得的分解 $\mathcal{T}^X$称为浅分解。
3. 运行 引理 5.5 中的算法，将分解 $\mathcal{T}^X_{\text{shallow}}$ 转换为所需的树分解 $\mathcal{T}^X=(T^X,\ 袋^X)$。

下面列出算法后续部分所需的本步骤的所有性质。性质 (P3) 此时可能还不太清楚，但它将在摊还运行时间的分析中发挥重要作用。

引理 5.6. 以下性质成立。

(P1) $\mathcal{T}^X$ 是宽度至多为 $6k+5$、高度至多为$\mathcal{O}(k+\log N)$的躯干图$_G(X)$ 的二叉树分解。

(P2) $V(T^X)\leqslant 2^{\mathcal{O}(k)}\cdot|T_{\text{pref}}|$。

(P3) 对于每个收集分量 $\mathfrak{C}\in\mathsf{Coll}(T_{\text{pref}},X)$，都存在一个叶节点 $t_{\mathfrak{C}}\in V(T^X)$，使得袋$^X(t_{\mathfrak{C}})=$ 接口$(\mathfrak{C})$。

(RT3) *Step 2* 的运行时间为 $2^{\mathcal{O}(k^3)}\cdot|T_{\text{pref}}|$。

证明. 我们证明该引理中的各个连续点。

命题 5.7。 $\mathcal{T}^X$ 是躯干图$_G$（X）的宽度至多为 $6k+5$ 且高度为$\mathcal{O}$（$k+\log N$）的二叉树分解。此外，V（T^X）$\leqslant 2^{\mathcal{O}(k)}\cdot|T_{\text{pref}}|$。

命题证明。回顾一下，由于 X 是一个 k-闭包，我们有 tw(躯干图$_G$（X）$\leqslant 2k+1$。因此，我们可以计算出宽度为 $\mathcal{T}^X{}_{\text{最优}}$ 的分解 $2k+1$。因此，分解$\mathcal{T}^X{}_{\text{浅}}$ 的宽度至多为 $3\cdot(2k+1)+2=6k+5$。这同样是 $\mathcal{T}^X$ 的宽度界，因为应用 引理 5.5 不会增加分解的宽度。

接下来，注意到 $|X|\leqslant\mathcal{O}(k^4\cdot|T_{\text{pref}}|)$，因为 X是 $\mathcal{O}(k^4)$-小的，这意味着 $|V(T^X{}_{\text{最优}})|\leqslant\mathcal{O}(k^4\cdot|T_{\text{pref}}|)$。由此可知，$\mathcal{T}^X{}_{\text{浅}}$的高度至多为 $\mathcal{O}(\log|X|)=\mathcal{O}(k+\log N)$。因此，根据引理 5.5，$\mathcal{T}^X$ 的高度确实至多为 $\mathcal{O}(k+\log N)$。

最后，对于 $\mathcal{T}^X$ 的规模，我们有 $|V(T^X)|\leqslant 2^{\mathcal{O}(k)}\cdot|V(T^X{}_{\text{浅}})|\leqslant 2^{\mathcal{O}(k)}\cdot|T^X{}_{\text{opt}}|\leqslant 2^{\mathcal{O}(k)}\cdot|袋(T_{\text{pref}})|$。 ◁

断言 5.8。 对于每个收集分量 $\mathfrak{C}\in$收集分量(T_{pref},X)，都存在一个叶节点 $t_{\mathfrak{C}}\in T^X$，使得袋$^X(t_{\mathfrak{C}})=$ 接口$(\mathfrak{C})$。

命题证明。考虑一个收集分量 $\mathfrak{C} \in \mathsf{Coll}(T_{\mathrm{pref}}, X)$。首先，注意到 $\mathsf{Interface}(\mathfrak{C})$ 是 $\mathsf{torso}_G(X)$ 中的一个团。事实上，如果 $\mathfrak{C}$ 未阻塞，那么根据引理 4.20，$\mathsf{Interface}(\mathfrak{C}) = N(\mathfrak{C})$ 是 $\mathsf{torso}_G(X)$ 中的一个团。如果 $\mathfrak{C}$ 受阻，那么 $\mathsf{Interface}(\mathfrak{C}) = \mathsf{bag}(t) \cap X$，其中 t 是 $\mathfrak{C}$ 的归属袋。然而，$\mathsf{bag}(t) \cap X$ 也是 $\mathsf{torso}_G(X)$ 中的一个团，因为 t 是一个阻塞。

因此，根据 事实 3.4，存在一个节点 $t \in V(T^X_{\text{浅}})$ 使得 $\text{接口}(\mathfrak{C}) \subseteq \text{袋}^X(t)$。然后，根据引理 5.5，我们知道存在一个叶节点 $t_{\mathfrak{C}} \in V(T^X)$ 使得 $\text{袋}^X(t_{\mathfrak{C}}) = \text{接口}(\mathfrak{C})$。 $\lhd$

命题 5.9。 步骤 *2* 的运行时间为 $2^{\mathcal{O}(k^3)} \cdot |T_{\mathrm{pref}}|$。

命题证明。 根据前述观察，计算 $\mathcal{T}^X_{\text{最优值}}$ 需要 $2^{\mathcal{O}(k^3)} \cdot |X|$ 时间，并且由于 $|X| \leqslant k^{\mathcal{O}(1)}|T_{\mathrm{pref}}|$，我们可以将该界写为 $2^{\mathcal{O}(k^3)} \cdot |T_{\mathrm{pref}}|$。另外两个步骤的运行时间可由 $2^{\mathcal{O}(k)} \cdot |X|$ 上界，因此命题得证。 $\lhd$

上述论断验证了所有必需的性质，因此证明完成。 □

步骤 3（拆分旧附属子树）

到目前为止，我们已经构造出了躯干图$_G(X)$的树分解 $\mathcal{T}^X$。现在，我们将为每个收集分量 $\mathfrak{C} \in \mathsf{Coll}(T_{\mathrm{pref}}, X)$ 构造一个树分解 $\mathcal{T}^{\mathfrak{C}}$，并将其附加到分解 $\mathcal{T}^X$ 上。具体而言，$\mathcal{T}^{\mathfrak{C}}$ 将是图 $G[\mathfrak{C} \cup \mathsf{Interface}(\mathfrak{C})]$的树分解，其根袋包含 $\mathsf{Interface}(\mathfrak{C})$。我们将首先描述该构造，然后证明其必需的性质，最后论证用于为所有 $\mathfrak{C} \in \mathsf{Coll}(T_{\mathrm{pref}}, X)$ 构造 $\mathcal{T}^{\mathfrak{C}}$ 的相关对象可以在时间 $k^{\mathcal{O}(1)} \cdot |F|$ 内计算出来。

现在我们描述 $\mathcal{T}^{\mathfrak{C}}$ 的构造。设 $\mathfrak{C}$ 是一个收集分量，并回顾 第 4.3 节 中归属袋的定义。我们称阻塞 $t \in \mathsf{Blockages}(T_{\mathrm{pref}}, X)$ 与 $\mathfrak{C}$ 相关联，如果以下任一条件成立：(1) $\mathfrak{C}$ 是一个受阻分量，且其归属袋 t 是；或 (2) $\mathfrak{C}$ 是一个未阻塞分量，且 $\mathfrak{C} \cap \mathsf{bag}(t)$ 非空。注意，阻塞和收集分量的定义意味着，每个阻塞至多与一个收集分量相关联。

现在，若 $\mathfrak{C}$ 是一个受阻连通分量，我们定义 $\mathcal{T}^{\mathfrak{C}} = (T^{\mathfrak{C}}, \text{袋}^{\mathfrak{C}})$，使得 $T^{\mathfrak{C}}$ 是以 $\mathfrak{C}$ 的归属袋为根的 T 子树的副本，而袋$^{\mathfrak{C}}$ 则类似地是该子树中袋函数的副本。若 $\mathfrak{C}$ 是一个未阻塞连通分量，则按如下方式构造 $\mathcal{T}^{\mathfrak{C}} = (T^{\mathfrak{C}}, \text{袋}^{\mathfrak{C}})$。对于一个阻塞$b \in$ 阻塞集(T_{pref}, X)，我们用 T_b 表示以 b 为根的 T 子树。我们用 $\mathsf{AB}(\mathfrak{C})$ 表示与 $\mathfrak{C}$ 相关联的阻塞集。然后，树 $T^{\mathfrak{C}}$ 定义如下

$$T^{\mathfrak{C}} = T[\{t \in F \mid \mathfrak{C} \cap \mathsf{bag}(t) \neq \emptyset\} \cup \bigcup_{b \in \mathsf{AB}(\mathfrak{C})} V(T_b)].$$

也就是说，$T^{\mathfrak{C}}$ 是 T 的子树，由以下部分组成：(1) 袋中包含 $\mathfrak{C}$ 中顶点的已探索节点；以及 (2) 以与 $\mathfrak{C}$ 相关联的阻塞集为根的子树。根据收集分量的定义，$T^{\mathfrak{C}}$ 是连通的；此外，$T^{\mathfrak{C}}$ 中唯一最高的节点（特别是 $T^{\mathfrak{C}}$ 的根，因为 $T^{\mathfrak{C}}$ 是连通的）对应于 $\mathfrak{C}$ 的归属袋。

我们注意到，$T^{\mathfrak{C}}$ 所有（受阻和未阻塞的）收集连通分量中的树 $\mathfrak{C}$ 都满足以下性质：

- 每个节点在 $V(T) \setminus F$中至多属于一棵树 $T^{\mathfrak{C}}$，
- 中每个节点 $F \setminus T_{\mathrm{pref}}$ 至多属于 $\ell + 1$ 棵树 $T^{\mathfrak{C}}$（因为 $F \setminus T_{\mathrm{pref}}$ 中的所有节点的袋规模都至多为 $\ell + 1$），并且
- T 中没有任何节点属于任意一棵树 $T^{\mathfrak{C}}$。

在本节余下部分中，对于节点 $t \in V(T^{\mathfrak{C}})$，我们用 origin($t$) 表示 T 中对应的节点。注意，origin 是从并集 $\bigcup_{\mathfrak{C}\in\mathsf{Coll}(T_{\mathrm{pref}},X)} V(T^{\mathfrak{C}})$ 到 $V(T)$ 的映射。

$\mathcal{T}^{\mathfrak{C}} = (T^{\mathfrak{C}},$ 未阻塞连通分量的袋$^{\mathfrak{C}}$)的袋$^{\mathfrak{C}}$(t)定义如下。对于原节点($T^{\mathfrak{C}}$)为 $\notin F$的节点$t \in V(T^{\mathfrak{C}})$（即，对于某个阻塞集 b，其原节点(t)为 $\in V(T_b)$），我们令 bag$^{\mathfrak{C}}$(t) =bag(origin$t$)。换言之，与 $\mathfrak{C}$ 相关联的阻塞集所根植的子树会从 $\mathcal{T}$原样复制到 $\mathcal{T}^{\mathfrak{C}}$。然后，对于原节点($t$)为 $\in F$的节点 $t \in V(T^{\mathfrak{C}})$，我们定义

$$\mathsf{pull}(t, \mathfrak{C}) = N(\mathfrak{C}) \cap (\mathtt{cmp}(\mathsf{origin}(t)) \setminus \mathsf{bag}(\mathsf{origin}(t))).$$

也就是说，pull(t, $\mathfrak{C}$)由 $N(\mathfrak{C})$中出现在原节点(t)下方的(T, 袋)子树各袋中的顶点组成，但不包括 bag(origin(t)) 中的顶点。最后，对于原节点(t)为 $\in F$ 的节点 $t \in T^{\mathfrak{C}}$，我们定义

$$\mathsf{bag}^{\mathfrak{C}}(t) = (\mathsf{bag}(\mathsf{origin}(t)) \cap N[\mathfrak{C}]) \cup \mathsf{pull}(t, \mathfrak{C}).$$

在袋$^{\mathfrak{C}}$(t)中加入 pull(t, $\mathfrak{C}$)，目的是确保 $N(\mathfrak{C})$位于 $\mathcal{T}^{\mathfrak{C}}$的根部：对于每个不在 $\mathcal{T}^{\mathfrak{C}}$的根袋中的 $v \in N(\mathfrak{C})$，我们将 v 加入到根与 $\mathcal{T}^{\mathfrak{C}}$ 中包含 v的最浅节点之间路径上的每个袋中。

至此，$\mathcal{T}^{\mathfrak{C}}$的定义就完成了。接下来，我们证明这一构造的一些基本性质。

引理 5.10. 设 $\mathfrak{C}$ 是一个收集分量。

(P4) $\mathcal{T}^{\mathfrak{C}}$ 是 $G[\mathfrak{C} \cup$接口$(\mathfrak{C})]$ 的有根二叉树分解 .

(P5) $\mathcal{T}^{\mathfrak{C}}$ 的根袋包含作为子集的 接口$(\mathfrak{C})$ 。

(P6) $T^{\mathfrak{C}}$ 的高度至多为 $\mathfrak{C}$ 的归属袋的高度。

证明. 首先，当 $\mathfrak{C}$ 是一个受阻连通分量时，令 $t \in V(T)$ 为 $\mathfrak{C}$ 的归属袋。根据以下事实，所有性质均直接成立：$\mathfrak{C} \subseteq$ cmp(t)、Interface($\mathfrak{C}$) $\subseteq$ bag(t)，以及 bag(t) $\subseteq \mathfrak{C} \cup$Interface($\mathfrak{C}$)。

接着，假设 $\mathfrak{C}$ 未受阻。回想一下，在这种情况下，Interface($\mathfrak{C}$) $= N(\mathfrak{C})$。首先，我们有 $T^{\mathfrak{C}}$，其袋限制在 $\mathfrak{C}$ 上，是 $G[\mathfrak{C}]$ 的树分解，因为 (T, 袋) 中顶点 $\mathfrak{C}$ 的所有出现都在 $T^{\mathfrak{C}}$ 中，并且构造这些袋时，我们从不移除顶点 $\mathfrak{C}$ 的出现。根据同样的论证，$\mathcal{T}^{\mathfrak{C}}$ 也覆盖 $\mathfrak{C}$ 与 $N(\mathfrak{C})$ 之间的所有边。然后，回想一下，根据引理 4.13，对于每个阻塞 b，cmp(b) $\cap X = \emptyset$，这意味着每个 $v \in N(\mathfrak{C})$ 都必须位于某个节点 $t \in V(T^{\mathfrak{C}})$ 的袋 bag(t) 中，且 origin(t) $\in F$。这意味着，即使在插入集合 pull($t, \mathfrak{C}$) 之后，$T^{\mathfrak{C}}$ 对顶点 $v \in N(\mathfrak{C})$ 也满足顶点条件。请注意，这些插入确保 $N(\mathfrak{C})$ = Interface($\mathfrak{C}$) 包含在 $\mathcal{T}^{\mathfrak{C}}$ 的根中，这表明，将袋限制在 $\mathfrak{C} \cup$Interface($\mathfrak{C}$) 上的 $\mathcal{T}^{\mathfrak{C}}$ 是 $G[\mathfrak{C} \cup$ Interface($\mathfrak{C}$)] 的树分解。

为了证明 $\mathcal{T}^{\mathfrak{C}}$ 的袋不包含 $\mathfrak{C} \cup$ Interface($\mathfrak{C}$) 之外的顶点，注意，根据未阻塞的收集分量的定义，与 $\mathfrak{C}$ 关联的每个阻塞集 b 都满足 bag(b) $\subseteq N[\mathfrak{C}]$ 且 cmp(b) $\subseteq \mathfrak{C}$。因此，阻塞集下方逐字复制的袋不包含 $V(G) \setminus N[\mathfrak{C}]$ 的任何元素；而 $\mathcal{T}^{\mathfrak{C}}$ 中已探索的袋已被明确截断至 $N[\mathfrak{C}]$。由此证明，$\mathcal{T}^{\mathfrak{C}}$ 是 $G[\mathfrak{C} \cup$Interface($\mathfrak{C}$)] 的树分解。$\mathcal{T}^{\mathfrak{C}}$ 是一棵二叉树，且 $T^{\mathfrak{C}}$ 的高度至多为 $\mathfrak{C}$ 的归属袋高度，这些事实均可直接由构造得出。

□

接下来，我们证明 $\mathcal{T}^{\mathfrak{C}}$ 的宽度至多为 ℓ。此外，袋的规模还满足一个更强的界，这对于界定势函数至关重要。这是我们最后一次使用 X 是 $d_{\mathcal{T}}$-极小的假设进行证明。

引理 5.11. 设 $t \in V(T^{\mathfrak{C}})$。

(P7) 若 *origin(t)* 已探索，则 $|bag^{\mathfrak{C}}(t)| < |bag(origin(t))|$ 且 $|origin^{-1}(origin(t))| \leqslant \ell + 1$。*(P8)* 若 *origin(t)* 未探索，则 $|bag^{\mathfrak{C}}(t)| = |bag(origin(t))|$ 且 $|origin^{-1}(origin(t))| = 1$。*(P9)* 有 $height_{T^{\mathfrak{C}}}(t) \leqslant height_T(origin(t))$。

证明。 高度性质 (P9) 显然成立，因为这来自构造。性质 (P8) 对于未探索节点，由构造以及每个阻塞至多与一个收集分量相关联这一事实可知成立。对于已探索节点， $|$原节点$^{-1}(t)| \leqslant \ell + 1$ 的性质 $t \in F \setminus T_{\text{pref}}$则由以下事实可知： 中的每个袋规模至多为 $F \setminus T_{\text{pref}}$，因此至多可以与 $\ell + 1$个不同的连通分量相交 $\ell + 1$ $\mathfrak{C}$。

剩下需要证明的是，当 origin(t) $\in F$ 时， $|\text{bag}^{\mathfrak{C}}(t)| < |\text{bag}(\text{origin}(t))|$ 。为此，我们需要利用 X 的 $d_{\mathcal{T}}$-极小性。注意，由于 origin(t) $\in F$，我们必然有 $\mathfrak{C}$ 未受阻。根据 bag$^{\mathfrak{C}}(t)$ 的定义，只需证明 $|\text{pull}(t, \mathfrak{C})| < |\text{bag}(\text{origin}(t)) \setminus N[\mathfrak{C}]|$。

首先，考虑 pull$(t, \mathfrak{C}) = \emptyset$ 的情况，此时需要证明 bag(origin(t)) 不是 $N[\mathfrak{C}]$ 的子集。在这种情况下，我们注意到，如果 bag(origin(t)) 是 $N[\mathfrak{C}]$ 的子集，那么根据引理 4.16，对于 $C \in \text{cc}(G - X)$，它将是 $N[C]$ 的子集；此时，如果 C 与 bag(origin(t)) 相交，它就会是一个分量阻塞；否则，它将是 $N(C)$ 的子集，因此是一个团阻塞，因为 C 与 bag(origin(t)) 不相交。

还需证明，如果 pull$(t, \mathfrak{C})$ 非空，那么 $|\text{pull}(t, \mathfrak{C})| < |\text{bag}(\text{origin}(t)) \setminus N[\mathfrak{C}]|$。为证明这一点，回顾第 3 节中函数 $d_{\mathcal{T}}$ 的定义：它是 $\mathcal{T}$ 的深度函数。为导出矛盾，假设 pull$(t, \mathfrak{C})$ 非空，且 $|\text{pull}(t, \mathfrak{C})| \geqslant |\text{bag}(\text{origin}(t)) \setminus N[\mathfrak{C}]|$。设 C 是 $G - X$ 的一个连通分量，使得 $N(\mathfrak{C}) = N(C)$（其存在性由引理 4.16 得出）。我们现在声称，

$$S := (N(C) \setminus \text{pull}(t, \mathfrak{C})) \cup (\text{bag}(\text{origin}(t)) \setminus N[\mathfrak{C}])$$

是一个 $(N(C), \text{bags}(T_{\text{pref}}))$-分隔集，这与 $N(C)$ 可通过袋 bags(T_{pref})-$d_{\mathcal{T}}$ 连通这一事实矛盾；根据引理 4.7，这又与 X 是 $d_{\mathcal{T}}$-极小的事实矛盾。首先注意，由于 pull$(t, \mathfrak{C}) \subseteq N(C)$，确实有 $|S| \leqslant |N(C)|$。此外，对于每个 $v \in \text{pull}(t, \mathfrak{C})$，包含 v 的最高层袋是 origin(t) 的真后代，因此对于所有 $v \in \text{pull}(t, \mathfrak{C})$ 和 $u \in \text{bag}(\text{origin}(t))$，都有 $d_{\mathcal{T}}(v) > d_{\mathcal{T}}(u)$，从而可得 $d_{\mathcal{T}}(S) < d_{\mathcal{T}}(N(C))$。还需证明 S 确实将 $N(C)$ 与 bags(T_{pref}) 分隔开来。为此，只需证明它将 pull$(t, \mathfrak{C})$ 与 bags(T_{pref}) 分隔开来，因为 $N(C) \setminus S = \text{pull}(t, \mathfrak{C})$。假设 P 是图 $G - S$ 中一条从 pull$(t, \mathfrak{C})$ 到 bags(T_{pref}) 的最短路径。由于 $\mathfrak{C}$ 与 bags(T_{pref}) 不相交，且 $N(\mathfrak{C}) \setminus S = \text{pull}(t, \mathfrak{C})$，因此 P 仅在其第一个顶点处与 $N[\mathfrak{C}]$ 相交。然而，注意到，包含 pull$(t, \mathfrak{C})$ 中顶点的袋的那些 $\mathcal{T}$ 都是 origin(t) 的真后代，且 origin(t) 是 T_{pref} 的某个附属节点的后代，因此 bag(origin(t)) 将 pull$(t, \mathfrak{C})$ 与 bags(T_{pref}) 分隔开来，故 P 必须与 bag(origin(t)) 相交。然而，由于 bag(origin(t)) 与 pull$(t, \mathfrak{C})$ 不相交， P 与 bag(origin(t)) 的交集必位于 bag(origin(t)) $\setminus N[\mathfrak{C}]$ 中，但 bag(origin(t)) $\setminus N[\mathfrak{C}] \subseteq S$，因此这样的路径 P 不可能存在，从而 S 是一个 $(N(C), \text{bags}(T_{\text{pref}}))$-分隔集。 □

作为下一步，我们将说明可以高效地构造这些分解。请注意，通过前缀重建更新进行这种构造，实际上就是为由原节点(F)中节点组成的子树显式构造($T^{\mathfrak{C}}$，袋$^{\mathfrak{C}}$)，并给出指针，说明应如何将以阻塞集为根的子树附加到此类节点上。特别地，稍微滥用一下记号，令 $\mathcal{T}^{\mathfrak{C}}|_F$ 表示 $\mathcal{T}^{\mathfrak{C}}$ 在原节点(t)为 $\in F$的节点 t 上的限制。

引理 5.12。 存在一种算法，给定 T_{pref}、 X、 F、 H和阻塞集$(T_{\mathrm{pref}},\ X)$，可计算一个长度为 |收集分量$(T_{\mathrm{pref}},\ X)$| 的元组列表：对于每个收集分量 $\mathfrak{C}$ ∈收集分量$(T_{\mathrm{pref}},\ X)$，该列表都包含一个由接口($\mathfrak{C}$)、高度($T^{\mathfrak{C}}$)以及以下内容组成的元组，

- 若 $\mathfrak{C}$ 未阻塞，则为 $\mathcal{T}^{\mathfrak{C}}$ 在原节点$^{-1}(F)$ 中的节点上的 $\mathcal{T}^{\mathfrak{C}}|_F$ 限制，以及从与 $\mathfrak{C}$ 相关联的阻塞到 $\mathcal{T}^{\mathfrak{C}}|_F$ 中应将其附加到的叶节点的映射，并且
- 若 $\mathfrak{C}$ 受阻，则为指向 $\mathfrak{C}$的归属袋的指针。

(RT4) 运行时间为 $k^{\mathcal{O}(1)} \cdot |F|$。

证明。 首先，受阻分量 $\mathfrak{C}$ 对应于 H 中满足 cmp(t) $\neq \emptyset$ 的孤立阻塞顶点 t 。可以使用 $\mathbb{D}^{\mathrm{aux}}$ 的 cmpsize 方法识别这些阻塞顶点。我们可以使用 $\mathbb{D}^{\mathrm{aux}}$ 的 cmpsize 方法测试 cmp(t) $\neq \emptyset$, 从而识别与受阻分量相关联的阻塞并将其输出。它们的高度 height($T^{\mathfrak{C}}$) 可从$\mathbb{D}^{\mathrm{aux}}$获得，并且对于它们，Interface($\mathfrak{C}$) $=$ bag(t) $\cap X$。

接下来，为构造未受阻分量的对象，我们首先进行预计算步骤，为每个节点 $t \in F \setminus T_{\mathrm{pref}}$ 计算集合 $X \cap$ cmp(t)。由于以下事实，这些集合可以自底向上方式在总运行时间 $k^{\mathcal{O}(1)} \cdot |F|$ 内计算：(1) 根据 c， $|X \cap \mathrm{cmp}(t)| \leqslant \mathcal{O}(k^4)$，这是因为 X的规模较小；(2) 根据引理 4.13，对于所有 $t \in$Blockages(T_{pref}, X)，均有 $X \cap$cmp(t) $= \emptyset$。我们还预先为每个顶点 $v \in$ bags($F \setminus T_{\mathrm{pref}}$)计算一个指向 $F \setminus T_{\mathrm{pref}}$ 中某个包含 v的袋的指针。

接下来，我们遍历探索图的连通分量 $C \in$ cc($H - X$)，这些分量对应于具有 $C \cap$ bags(F) $\neq \emptyset$ 的节点，即对应于未阻塞的收集分量。现在固定一个连通分量 $C \in$ cc($H - X$)。注意， C 唯一标识一个未阻塞的收集分量$\xi^{-1}(C) = \mathfrak{C}$，并且注意 $N_H(C) = N(\mathfrak{C})$。现在， $\mathcal{T}^{\mathfrak{C}}|_F$中的节点可识别为集合$\{t \in F \mid \mathrm{bag}(t) \cap C \neq \emptyset\}$，利用先前计算出的指针，可以在时间 $k^{\mathcal{O}(1)} \cdot |\mathcal{T}^{\mathfrak{C}}|_F|$ 内计算它们，因为它们对应于 T 的一个连通子树。集合 $N(\mathfrak{C})$ 可以在时间 $k^{\mathcal{O}(1)} \cdot |C|$ 内计算，因为 $|N(\mathfrak{C})| \leqslant \mathcal{O}(k)$。然后，对所有 $t \in \mathcal{T}^{\mathfrak{C}}|_F$，利用预先计算的集合 $X \cap$ cmp(t)，可以在时间 $k^{\mathcal{O}(1)} \cdot |\mathcal{T}^{\mathfrak{C}}|_F|$ 内计算集合 pull($t, \mathfrak{C}$)，因为 pull($t, \mathfrak{C}$) $= N(\mathfrak{C}) \cap (X \cap$cmp(origin($t$))) $\setminus$ bag(origin(t))。集合 bag(origin(t)) $\cap N[\mathfrak{C}]$ 可在时间 $k^{\mathcal{O}(1)} \cdot |\mathcal{T}^{\mathfrak{C}}|_F|$ 内直接计算。因此，所有袋 bag$^{\mathfrak{C}}$(t) 也可在时间 $k^{\mathcal{O}(1)} \cdot |\mathcal{T}^{\mathfrak{C}}|_F|$ 内计算。至此， $\mathcal{T}^{\mathfrak{C}}|_F$ 的构造完成。随后，从与 $\mathfrak{C}$ 相关联的阻塞集指向 $\mathcal{T}^{\mathfrak{C}}|_F$ 的叶节点的指针，可在时间 $k^{\mathcal{O}(1)} \cdot |F|$ 内计算，因为这些阻塞集恰好是 $C \cap$ Blockages(T_{pref}, X)，且其父节点位于 $\mathcal{T}^{\mathfrak{C}}|_F$ 中。高度 height($T^{\mathfrak{C}}$) 可在时间 $\mathcal{O}(|\mathcal{T}^{\mathfrak{C}}|_F|)$ 内计算：使用 $\mathbb{D}^{\mathrm{aux}}$ 计算与 $\mathfrak{C}$ 相关联的阻塞集的高度，并应用归纳法。

总而言之，对于每个收集分量 $\mathfrak{C}$，我们花费 $k^{\mathcal{O}(1)} \cdot (|C| + |V(T^{\mathfrak{C}}|_F)|)$ 的时间来构造 $\mathcal{T}^{\mathfrak{C}}|_F$和指针。$C$ 的规模总和至多为 $|V(H)| \leqslant \mathcal{O}(k \cdot |F|)$，而$V(T^{\mathfrak{C}}|_F)$的规模总和至多为 $\mathcal{O}(k \cdot |F|)$，因此总运行时间为 $k^{\mathcal{O}(1)} \cdot |F|$。 □

步骤④（附加辅助子树）

此时，我们已经构造出躯干图$_G(X)$的树分解 $\mathcal{T}^X$ ，以及每个收集的连通分量 $\mathfrak{C}$对应的 $G[\mathfrak{C} \cup$ Interface($\mathfrak{C}$)树分解 $\mathcal{T}^{\mathfrak{C}}$] 。我们知道， $T^{\mathfrak{C}}$的根包含 Interface($\mathfrak{C}$)（性质 (P5)），并且存在一个叶节点 $t_{\mathfrak{C}} \in V(T^X)$，使得袋$^X(t_{\mathfrak{C}})$ =Interface($\mathfrak{C}$)（性质 (P3)）。因此，一个自然的想法是：将每棵树 $T^{\mathfrak{C}}$的根与相应的顶点 $t_{\mathfrak{C}} \in V(T^X)$用一条边连接起来，从而构造出最终的分解 $\mathcal{T}'$ 。

遗憾的是，许多收集的连通分量可能具有相同的接口 $B \subseteq X$，若将它们的根连接到 T^X 的同一个顶点，就会破坏数据结构所维护的二叉树分解不变量。一个看似可行的变通办法是：构造一棵以 $t_{\mathfrak{C}}$ 为根、规模足够大的完全二叉树 T^{bin}，并使其每个袋都包含 Interface($\mathfrak{C}$)。然后，我们就可以将每棵树 $T^{\mathfrak{C}}$ 分别接到 T的不同叶节点上二叉。

然而，这种方法存在一个微妙的问题：构造 T^{bin} 会增加所得树分解的势值；如果许多收集的连通分量共享同一个接口，势值的增加可能相当显著。因此，我们必须确保构造 T^{bin} 的摊销成本不超过创建收集的连通分量所带来的势值下降。粗略地说，只有当最终树分解中 T^{bin} 的顶点高度之和至多与树 $T^{\mathfrak{C}}$的高度之和成正比时，才能做到这一点。遗憾的是，完全二叉树并不总是满足这一条件。

然而，我们将借鉴一种类似哈夫曼编码的思路来解决这个问题，具体如下。直观地说，树 T^{bin} 应当采用倾斜结构，以满足以下性质：给定两个具有相同接口的收集分量的树分解 $T^{\mathfrak{C}_1}, T^{\mathfrak{C}_2}$，如果 $T^{\mathfrak{C}_1}$的高度远大于 $T^{\mathfrak{C}_2}$，那么 $T^{\mathfrak{C}_1}$应附接到 T^{bin}中比 $T^{\mathfrak{C}_2}$ 所附接到的叶节点更靠近 T^{bin}的根的叶节点上。下面的引理将这一直觉形式化。

引理 5.13。设 $h_1, \ldots, h_m$ 为正整数，并且 $Q := h_1 + \ldots + h_m$。那么，存在一棵高度为 $\mathcal{O}(\log Q)$ 的有根二叉树 T，其叶节点按某种顺序标记为 $h_1, \ldots, h_m$，使得 $\sum_{v \in V(T)} \text{叶高}(v) \leqslant 26Q$，其中 叶高 定义如下：

$$\mathsf{lheight}(v) := \begin{cases} \mathsf{label}(v) & \textit{if } v \textit{ is a leaf,} \\ 1 + \max\{\mathsf{lheight}(c) : c \textit{ is a child of } v\} & \textit{otherwise.} \end{cases}$$

此外，这样的树可以在 $\mathcal{O}(m + \log Q)$ 时间内计算得到。

证明。设 $C\colon \mathbb{Z}_{\geqslant 1} \to \mathbb{Z}_{\geqslant 0}$ 是由公式 $C(a) := \lceil \log_2 a \rceil$ 给出的函数。换言之，$C(a)$ 是满足 $2^{C(a)} \geqslant a$ 的最小非负整数。特别地，我们有 $2^{C(a)} < 2a$。

将 $h_1, \ldots, h_m$ 划分为 $G_0, \ldots, G_{C(Q)}$ 个组，具体如下：如果 $h_i \in (\frac{Q}{2^{j+1}}, \frac{Q}{2^j}]$，则将 h_i 放入~~G_j~~。这是一个定义良好的划分，因为对于 $j = 0, \ldots, C(Q)$，这些区间彼此不相交，并且它们的并集覆盖区间 $[1, Q]$。

现在，对于每个非空组 G_i，我们按如下方式构造一棵有根二叉树 T_i。设 G_i 的元素为 g_1、……、g_a。然后，令 T_i 为一棵任意的有根二叉树，其 a 个叶节点分别标记为 g_1、……、g_a，且所有叶节点到根的距离至多为 $C(a)$。由 $2^{C(a)} \geqslant a$ 可知，这样的树存在。如果 G_i 为空，我们也假设 T_i 为空。

下一步，构造一条路径 $P = v_0$、……、$v_{C(Q)}$，其根为 v_0。对于每个 $i = 0$、……、$C(Q)$，如果 G_i 非空，则将 T_i 的根设为 v_i 的子节点。移除 P 中没有连接任何子树的一个后缀。至此，所需的 T 已描述完毕。容易看出，该构造可在 $\mathcal{O}(m + \log Q)$ 时间内完成。

剩下要证明的是 T 所需的性质。由于每个 T_i 的高度为 $\mathcal{O}(\log Q)$，且路径 P 的长度为 $\mathcal{O}(\log Q)$，因此显然 T 的高度也为 $\mathcal{O}(\log Q)$。

接下来，我们将证明在一个特定的 T_i中，v 的 lheight(v) 之和的一个界。设G_i的元素为 g_1、……、$g_a \in (\frac{Q}{2^{i+1}}, \frac{Q}{2^i}]$。我们按顶点 $v \in V(T_i)$到以 v为根的子树中最远叶节点的距离 j对其分组。对于 $j = 0$，该组由 T_i的叶节点组成。这些叶节点按 G_i编号，因此它们的 lheight 值不超过$\frac{Q}{2^i}$。接下来，距离最远叶节点 $j \geqslant 1$ 的顶点，其 lheight 值至多为 $\frac{Q}{2^i} + j$（由 lheight 的定义可知），并且这类顶点至多有v个（因为在 g_1中的深度至多为 $v(T_i)\, G_i$）。因此，我们得到以下界：

lheight 的定义）且这类顶点至多有 $2^{C(a)-j}$ 个（由每个此类顶点在 T_i中的深度至多为 $C(a)$ $-j$ 这一事实可知）。因此，我们得到以下界

$$\sum_{v\in V(T_i)} \mathsf{lheight}(v) \leqslant \sum_{j=0}^{C(a)} 2^{C(a)-j}\left(\frac{Q}{2^i}+j\right) = 2^{C(a)}\left(\frac{Q}{2^i}\sum_{j=0}^{C(a)} 2^{-j} + \sum_{j=0}^{C(a)} j\cdot 2^{-j}\right) \leqslant$$
$$\leqslant 2^{C(a)}\left(\frac{Q}{2^i}\sum_{j=0}^{\infty} 2^{-j} + \sum_{j=0}^{\infty} j\cdot 2^{-j}\right) = 2^{C(a)}\left(2\cdot\frac{Q}{2^i}+2\right).$$

作为补充条件 $2^{C(a)}\cdot 2 \leqslant 4a$ 和 $2^{C(a)}\cdot 2\cdot \frac{Q}{2^i} \leqslant 8a\cdot \frac{Q}{2^{i+1}} < 8\,(g_1+\ldots+g_a)$，我们发现 at

$$\sum_{v\in V(T_i)} \mathsf{lheight}(v) \leqslant 8(g_1+\ldots+g_a)+4a \leqslant 12(g_1+\ldots+g_a). \tag{2}$$

对 公式 (2) 在所有 $i=0$、1、.....、C（Q）求和，可得

$$\sum_{i=0}^{C(Q)} \sum_{v\in V(T_i)} \mathsf{lheight}(v) \leqslant 12Q.$$

剩下要做的是界定 P 中各顶点的 lheight 之和。定义

$$l_i := \frac{Q}{2^i} + 2C(Q) - i + 4.$$

现在我们通过对 $i=C$（Q） C（Q）-1、.....、0 进行归纳法，证明 lheight(v_i) $\leqslant l_i$。每个顶点 v_i 至多有两个子节点：

- T_i 的根 r_i（如果 G_i 非空）：由于 $|G_i| \leqslant 2^{i+1}$，我们得到 lheight(r_i) $\leqslant \frac{Q}{2^{i+1}} + (i+1)$ $\leqslant l_i - 1$;
- v_{i+1} 若 v_{i+1} 存在：则有 lheight(v_{i+1}) $\leqslant l_{i+1} \leqslant l_i - 1$。

因此，确实有 lheight(v_i) $\leqslant l_i$。所以，

$$\sum_{v\in P} \mathsf{lheight}(v) \leqslant \sum_{i=0}^{C(Q)} l_i \leqslant \sum_{i=0}^{C(Q)} \frac{Q}{2^i} + (C(Q)+1)(2C(Q)+4) \leqslant 2Q + (C(Q)+1)(2C(Q)+4).$$

现在，由于 $C(Q) < \log_2 Q + 1$，我们得到 $(C(Q)+1)(2C(Q)+4) < (\log_2 Q + 2)(2\log_2 Q + 6)$。运用实分析中的标准工具，可以证明对于所有 $Q \in \mathbb{Z}_{\geqslant 1}$，都有

$$(\log_2 Q + 2)(2\log_2 Q + 6) \leqslant 12Q.$$

因此，

$$\sum_{v\in P} \mathsf{lheight}(v) \leqslant 14Q.$$

我们得出结论：

$$\sum_{v\in V(T)} \mathsf{lheight}(v) = \sum_{i=0}^{C(Q)} \sum_{v\in V(T_i)} \mathsf{lheight}(v) + \sum_{v\in P} \mathsf{lheight}(v) \leqslant 12Q + 14Q = 26Q. \qquad \square$$

证明了引理 5.13 后，我们可以定义将附加到已构造分解 $\mathcal{T}^X$ 上的树分解。令 $\mathcal{I}(T_{\text{pref}}, X) := \{\text{Interface}(\mathfrak{C}) \mid \mathfrak{C} \in \text{Coll}(T_{\text{pref}}, X)\}$ 表示收集分量的所有接口构成的集合。从引理 5.12 得到表示收集分量的元组后，我们按接口对这些元组分组，也就是说，遍历所有集合 $B \in \mathcal{I}(T_{\text{pref}}, X)$，并列出接口为 B 的所有连通分量 $\mathfrak{C}_1, \mathfrak{C}_2, \ldots, \mathfrak{C}_m \in \text{Interface}^{-1}(B)$。使用桶排序，通过标准论证可在 $|\text{Coll}(T_{\text{pref}}, X)| \cdot k^{\mathcal{O}(1)}$ 时间内完成此操作。

现在，固定 $B \in \mathcal{I}(T_{\text{pref}}, X)$，考虑连通分量 $\mathfrak{C}_1, \mathfrak{C}_2, \ldots, \mathfrak{C}_m \in \text{Interface}^{-1}(B)$，并令 $h_i = \text{height}(T^{\mathfrak{C}_i}) + 1$，其中 $i \in \{1, \ldots, m\}$。我们对整数 $h_1, \ldots, h_m$ 运行引理 5.13 中的算法，得到一棵树 T^B_{pref}，其叶节点标记为整数 h_i。接着，我们的构造是：将树 $T^{\mathfrak{C}_i}$ 附加到 T^B pref，对于 $i = 1, \ldots, m$，方法是将 $T^{\mathfrak{C}_i}$ 的根作为 T^Bpref 中标记为 h_i 的叶节点的子节点，从而使 T^Bpref 的每个叶节点恰好有一个子节点。令 T^B 为所得的二叉树，并注意，对于 T^B 中满足以下条件的每个节点 t_i：它是 T^B_{pref} 中附加了 $T^{\mathfrak{C}_i}$ 的叶节点，$\text{height}_{T^B}(t_i) = h_i$。

要定义树分解 $\mathcal{T}^B = (T^B, \text{袋}^B)$，还需定义函数 bag^B。我们令 $\text{bag}^B|_{V(T^{\mathfrak{C}_i})} = \text{bag}^{\mathfrak{C}_i}$，并且对每个 $t \in V(T^B_{\text{pref}})$，令 $\text{bag}^B(t) = B$。

现在，让我们分析这个过程。首先，对于 T_{pref} 的每个附属节点 t，我们证明，包含在 $\text{cmp}(t)$ 中的所有收集分量的所有可能接口数量，都有一个仅取决于 k 的上界。注意，这个上界正是我们要求闭包 X 为 c-小的原因。

引理 5.14. 对于每个附属节点 $t \in \text{App}(T_{\text{pref}})$，我们有

$$|\{\text{Interface}(\mathfrak{C}) \mid \mathfrak{C} \in \text{Coll}(T_{\text{pref}}, X), \mathfrak{C} \subseteq \text{cmp}(t)\}| \leqslant k^{\mathcal{O}(k)}.$$

证明。设 $\mathfrak{C} \in \text{Coll}(T_{\text{pref}}, X)$ 满足 $\mathfrak{C} \subseteq \text{cmp}(t)$。回顾一下，$\text{Interface}(\mathfrak{C}) \subseteq X$，这意味着 $\text{Interface}(\mathfrak{C}) \subseteq X \cap (\text{cmp}(t) \cup \text{adh}(t))$。由于 X 是 c-小的，我们有 $|X \cap \text{cmp}(t)| \leqslant c \in \mathcal{O}(k^4)$。此外，我们有 $|X \cap \text{adh}(t)| \leqslant |\text{bag}(t)| \leqslant \ell + 1$。由于 $|\text{Interface}(\mathfrak{C})| \leqslant 2k + 2$ ($\text{Interface}(\mathfrak{C})$ 是躯干图 $\text{torso}_G(X)$ 中的一个团，且 $\text{tw}(\text{torso}_G(X)) \leqslant 2k + 1$)，因此对于满足 $\mathfrak{C} \subseteq \text{cmp}(t)$ 的 $\text{Interface}(\mathfrak{C})$，至多有 $\sum_{i=0}^{2k+2} \binom{c+\ell+1}{i} = (k^4)^{\mathcal{O}(k)} = k^{\mathcal{O}(k)}$ 种可能取值。 □

我们立即得到图中所有收集分量的接口总数的一个界：

推论 5.15. 以下不等式成立：

$$|\mathcal{I}(T_{\text{pref}}, X)| \leqslant k^{\mathcal{O}(k)} \cdot |T_{\text{pref}}|.$$

证明. 对于每个收集分量 $\mathfrak{C}$，都存在唯一的附属节点 t，使得 $\mathfrak{C} \subseteq \text{cmp}(t)$。由于 $(T, \text{袋})$ 是二叉树分解，我们知道 $|\text{App}(T_{\text{pref}})| \leqslant |T_{\text{pref}}| + 1$。因此，$|\mathcal{I}(T_{\text{pref}}, X)| \leqslant k^{\mathcal{O}(k)} \cdot |T_{\text{pref}}|$，证毕。 □

我们总结一下这一步细化操作中构造的性质 n.

引理 5.16. 设 $B \in \mathcal{I}(T_{\text{pref}}, X)$。
(P10) $\mathcal{T}^B = (T^B, \text{袋}^B)$ 是 $G[B \cup \bigcup_{\mathfrak{C} \in \text{接口}^{-1}(B)} \mathfrak{C}]$ 的二叉树分解，其宽度至多为 ℓ。

(P11) T^B 的高度至多为

$$\mathcal{O}(\log N) + \max_{\mathfrak{C}\in \mathsf{Interface}^{-1}(B)} \mathsf{height}(\mathcal{T}^{\mathfrak{C}}).$$

(P12) $\mathcal{T}^B$ 的根袋恰好包含集合 B。

(P13) V 中节点的高度之和满足以下界：

$$\sum_{t\in V(T^B_{\mathrm{pref}})} \mathsf{height}_{T^B}(t) \leqslant 52 \cdot \sum_{\mathfrak{C}\in \mathsf{Interface}^{-1}(B)} \mathsf{height}(T^{\mathfrak{C}}).$$

证明。首先，我们证明 $\mathcal{T}^B$ 确实是一个有效的树分解。$\mathcal{T}^B$ 是有效树分解 $\mathcal{T}^{\mathfrak{C}}$ 与前缀 T^B_{pref} 的串接，而后者事实上是 $G[B]$的一个有效树分解。回顾一下，所有收集的连通分量都是顶点不相交的。此外，边条件也得到满足：对于一条边 uv，若其端点为 u、$v \in B$，则边条件是显然的；如果一条边 uv的某个端点（例如 v）属于收集的连通分量 $\mathfrak{C}$，则 $u \in N[\mathfrak{C}] \subseteq \mathfrak{C} \cup \mathsf{Interface}(\mathfrak{C})$。由于 $\mathcal{T}^{\mathfrak{C}}$ 是 $G[\mathfrak{C} \cup \mathsf{Interface}(\mathfrak{C})]$ 的树分解（性质 (P4)），我们推知 u和 v共同出现在 $\mathcal{T}^{\mathfrak{C}}$的某个袋中。因此，仍需说明每个顶点 $v \in B$ 都出现在 $\mathcal{T}^B$的节点的一个连通子集中。这一点立即由以下事实得出：T^B_{pref} 在所有袋中都包含 B，并且对于每个收集的连通分量 $\mathfrak{C}$，$\mathcal{T}^{\mathfrak{C}}$ 的根袋包含 $\mathsf{Interface}(\mathfrak{C}) = B$ 作为子集（性质 (P5)）。

由引理 5.13可知，T^B_{pref}的高度至多为 $\mathcal{O}(\log \sum_{\mathfrak{C}\in 接口^{-1}(B)}(\mathsf{height}(T^{\mathfrak{C}}) + 1))$；由于 $\mathsf{height}(T^{\mathfrak{C}}) \leqslant N$且 $|\mathsf{Interface}^{-1}(B)| \leqslant N$，因此该值至多为 $\mathcal{O}(\log(N^2)) = \mathcal{O}(\log N)$。根据构造可得，$T^B$的高度至多为 $\mathcal{O}(\log N) + \max_{\mathfrak{C}\in \mathsf{Interface}^{-1}(B)} \mathsf{height}(T^{\mathfrak{C}})$。

由于 T^B_{pref} 是一棵非空树，其根袋（同时也是 $\mathcal{T}^B$ 的根袋）恰好包含集合 B。

为了界定 $V(T^B\mathsf{pref})$ 中节点高度之和，我们注意到：在 $T^B\mathsf{pref}$ 中，$\mathcal{T}^{\mathfrak{C}_i}$所连接到的叶节点 $t_i \in V(T^B\mathsf{pref})$ 的高度 $\mathsf{height}_{T^B}(t_i)$ 恰好为 $\mathsf{height}(T^{\mathfrak{C}_i}) + 1 = h_i$，因此 $V(T^B\mathsf{pref})$ 中的高度恰好对应于引理 5.13 陈述中的 lheight。因此，由引理 5.13 可得

$$\sum_{t\in V(T^B_{\mathrm{pref}})} \mathsf{height}_{T^B}(t) \leqslant 26 \cdot \sum_{\mathfrak{C}\in \mathsf{Interface}^{-1}(B)} (\mathsf{height}(T^{\mathfrak{C}}) + 1) \leqslant 52 \cdot \sum_{\mathfrak{C}\in \mathsf{Interface}^{-1}(B)} \mathsf{height}(T^{\mathfrak{C}}).$$

□

最后，我们给出这一步的运行时间界。

引理 5.17. 存在一种算法，给定引理 *5.12*的输出，可以计算集合 $\mathcal{I}(T_{\mathrm{pref}}, X)$，并且对于每个 $B \in \mathcal{I}(T_{\mathrm{pref}}, X)$，计算树 $T^B{}_{\mathrm{pref}}$以及从已收集分量的表示 $\mathfrak{C} \in 接口^{-1}(B)$ 到 T^B_{pref} 中相应叶节点的指针；构造过程中，树分解 $\mathcal{T}^{\mathfrak{C}}$ 附接到这些叶节点上。

(RT5) 运行时间为 $|F| \cdot k^{\mathcal{O}(1)} + |T_{\mathrm{pref}}| \cdot k^{\mathcal{O}(k)} \cdot \log N$。

证明。 注意到 引理 5.12 的输出规模至多为 $|F| \cdot k^{\mathcal{O}(1)}$。我们首先使用桶排序对接口$B$ =接口$(\mathfrak{C})$ 进行分组，这需要 $\mathsf{Coll}(T_{\mathrm{pref}}, X) \cdot k^{\mathcal{O}(1)} = k^{\mathcal{O}(1)} \cdot |F|$ 时间。

接下来，剩余运行时间显然由调用引理 5.13的总运行时间所主导。对一个集合 $B \in \mathcal{I}(T_{\text{pref}}, X)$ 单次应用该引理需要的时间为

$$\mathcal{O}(|\mathsf{Interface}^{-1}(B)| + \log N)$$

因为所有收集的连通分量的高度之和可以界定为 $\mathcal{O}(N^2)$。

因此，由于所有 $|\text{接口}^{-1}(B)|$ 的 B 总和可以界定为 $|F|k^{\mathcal{O}(1)}$，只需考虑 引理 5.12输出的规模，以及 $|T_{\text{pref}}| \cdot k^{\mathcal{O}(k)}$根据推论 5.15，总运行时间可以界定为 $|F| \cdot k^{\mathcal{O}(1)} + |T_{\text{pref}}| \cdot k^{\mathcal{O}(k)} \cdot \log N$。 □

步骤 5 (将分解的各个部分连接起来)
现在来定义最终的分解 $\mathcal{T}' = (T', \text{袋}')$。在上一步中，我们针对每个 $B \in \mathcal{I}(T_{\text{pref}}, X)$ 定义了一个二叉树分解 $\mathcal{T}^B$，它包含所有收集的连通分量 $\mathfrak{C}$，使得 $\mathsf{Interface}(\mathfrak{C}) = B$。此外，$\mathcal{T}^B$ 的根袋恰好包含集合 B （性质 (P12)），并且在步骤 2 中构造的树分解 $\mathcal{T}^X$ 中存在一个叶袋 $t_B \in V(T^X)$，使得袋$^X(t_B) = B$ （性质 (P3)）。

因此，要构造 T'，只需对每个 $B \in \mathcal{I}$（T_{pref}， X），将T^B的根与相应的顶点 $t_B \in V$（T^X）用一条边连接起来。函数袋′就是函数袋^{X}与袋^{B}的并集，其中 $B \in \mathcal{I}$（T_{pref}， X）。显然，给定 $\mathcal{T}^X$ 以及引理 5.12 和 5.17 的输出，我们可以在 $|F| \cdot 2^{\mathcal{O}(k)}$时间内构造一个规模至多为$|F| \cdot 2^{\mathcal{O}(k)}$ 的前缀重建更新描述，将 $\mathcal{T}$变为 $\mathcal{T}'$。

(RT6) 步骤 5 的运行时间为 $|F| \cdot 2^{\mathcal{O}(k)}$。

此时，我们应当证明该过程的正确性。势函数 Φ 的精确公式以及摊还运行时间的分析将在下一节给出。

引理 5.18。 $(T', \text{袋}')$ 是 G 的一个有效树分解，其宽度至多为 ℓ。

证明。 首先，回顾一下，$(T', \text{袋}')$ 是通过粘合树分解 $\mathcal{T}^X$ 和 $\mathcal{T}^B$得到的，对每个 $B \in \mathcal{I}(T_{\text{pref}}, X)$均如此。这些分解的宽度都至多为 ℓ（性质(P1) 和(P10)），因此$(T', \text{袋}')$ 的宽度也至多为 ℓ。

还需证明 $(T', \text{袋}')$ 确实是 G 的一个有效树分解。根据树分解的定义，我们需要验证两个事实。

命题 5.19。 对于每个顶点 $v \in V(G)$，节点子集 $\{t \in V(T') : v \in \text{袋}'(t)\}$ 在 T' 中诱导出一棵非空连通子树。

命题证明。考虑一个顶点 $v \in V(G)$。

- 如果 $v \notin X$，则存在唯一的收集分量 $\mathfrak{C}$，使得 $v \in \mathfrak{C}$。由于 $\mathcal{T}^{\mathfrak{C}}$ 是 $G[\mathfrak{C} \cup \text{接口}(\mathfrak{C})]$，的一个有效树分解，$T^{\mathfrak{C}}$ 中包含 v 的节点子集必须在 $T^{\mathfrak{C}}$ 中诱导出一棵连通子树。此外，根据 $\mathcal{T}'$ 的构造，我们可以得出结论： v 不可能出现在 $\mathcal{T}'$ 中 $T^{\mathfrak{C}}$ 之外的任何袋中。

- 现在，假设 $v \in X$。顶点 v 出现在 $\mathcal{T}^X$ 的一个连通节点子集中，并且对于每个满足 $x \in B$ 的 $B \in \mathcal{I}$（$T_{\text{pref}}\, X$），它也出现在树 $\mathcal{T}^B$中节点的连通子树里。此外，根据$\mathcal{T}'$的构造，树 T^B 通过连接两个恰好包含集合 B的袋，附接到树 T^X 上。因此， v 必定出现在 $\mathcal{T}'$的一个连通节点子集中。

◁

断言 5.20. 对于每条边 $uv \in E(G)$，都存在一个节点 $t \in V(T')$，使得 $\{u, v\} \subseteq$ 袋′(t)。

命题证明。考虑一条边 $uv \in E(G)$。

- 如果 $\{u, v\} \subseteq X$，那么 $uv \subseteq E($躯干图$_G(X))$，因此必定存在一个袋 t 位于前缀 T^X中，且同时包含 u 和 v，因为 $\mathcal{T}^X$ 是躯干图$_G(X)$ 的树分解(性质(P1))。
- 如果 $\{u, v\} \cap X = \emptyset$，则存在唯一一个连通分量 $\mathfrak{C} \in$ Coll(Tpref X)同时包含 u 和v（因为 $uv \in E(G)$，且不同的收集连通分量之间没有边）。令$B =$ Interface(𝔠)。那么 $\mathcal{T}$包含树分解 $\mathcal{T}^B$ 作为其子树。由于 $\mathcal{T}^B$ 是 $G[B \cup$Interface$^{-1}(B)]$ 的树分解（性质(P10)），因此 u 和 v 同时出现在 $\mathcal{T}^B$的某个袋中。
- 最后，如果顶点 u 和 v 中恰有一个属于 X，设其为 $u \in X$，那么存在唯一一个连通分量 $\mathfrak{C} \in$ Coll(Tpref X)包含 v。同样，令 $B =$Interface(𝔠)。根据接口的定义，我们有 $u \in B$。同样，由于 $\mathcal{T}^B$ 是 $G[B \cup$Interface$^{-1}(B)]$ 的树分解（性质(P10)），因此 u和 v 同时出现在 $\mathcal{T}^B$的某个袋中。

◁

主张 5.19 和 5.20 完成了该引理的证明。 □

5.3 摊还运行时间分析

本节给出(RT1) 和 (POT)这两个已定义细化操$\widetilde{\ }$作性质的证明。我们保留第 5.2 节中的所有记号。此外，记 $F := F \setminus T_{\text{pref}}$ 为所有已探索节点的集合，但不包括 T_{pref}中的节点。

将细化操作连续步骤的运行时间 (RT2) – (RT6) 相加后，我们立即得到以下结论。

事实 5.21。 调用refine(T_{pref}) 的最坏情况时间复杂度上界为

$$2^{\mathcal{O}(k^9)}(|F| + |T_{\text{pref}}| \cdot \log N) \leqslant 2^{\mathcal{O}(k^9)}(|\widetilde{F}| + |T_{\text{pref}}| \cdot \log N).$$

注意，在数据结构运行过程中，我们可以轻松控制 $|T_{\text{pref}}|$ 和 log N这两个表达式。遗憾的是，已探索区域的规模 $|F|$ 可能任意大；在最坏情况下，探索 F可能包含 T的所有节点。因此，为了证明算法运行时间满足所需的界，我们需要将势函数的下降幅度限制在$\widetilde{\ }$探索 F 的规模范围内。

在继续分析摊还运行时间之前，我们还需要证明一个辅助引理。

引理 5.22。 存在一个映射

$$\mathsf{collmap} : \mathsf{Coll}(T_{\text{pref}}, X) \to \widetilde{F} \cup \mathtt{App}(T_{\text{pref}})$$

满足以下性质：

(i) 对每个 $\mathfrak{C} \in$ *Coll*(T_{pref}, X)，都有 *height*$(\mathfrak{C}) \leqslant$ *height*$_T$*(collmap*$(\mathfrak{C}))$；*(ii)* 对每个 $\mathfrak{C} \in$ *Coll*(T_{pref}, X)，若 *collmap*$(\mathfrak{C}) \notin$ *App*(T_{pref})，则 *weight*$(\mathfrak{C}) <$ *|bag(collmap*$(\mathfrak{C}))|$；*(iii)* 对每个 $t \in \widetilde{F} \cup$ *App*(T_{pref})，都有 $|$*collmap*$^{-1}$*(t)*$| \leqslant \ell + 4$。

证明。 我们如下定义 collmap。取 $\mathfrak{C} \in \mathrm{Coll}(T_{\mathrm{pref}},\ X)$。令 t 为 $\mathfrak{C}$ 的归属袋。于是：

- 如果 $\mathfrak{C}$ 未阻塞，则设置 $\mathrm{collmap}(t) := t$。这里回顾一下，t 是 T 中满足 $\mathrm{bag}(t)$包含 $\mathfrak{C}$的已探索顶点的最浅节点。
- 如果 $\mathfrak{C}$ 受阻，则当 $t \in \mathrm{App}(T_{\mathrm{pref}})$时设置 $\mathrm{collmap}(t) := t$；否则设置 $\mathrm{collmap}(t) := \mathrm{parent}(t)$。这里回顾一下，$t$ 是 $\xi(\mathfrak{C})$中唯一的阻塞。

回顾一下，$\mathrm{height}(\mathfrak{C}) = \mathrm{height}_T(t)$，$\mathrm{weight}(\mathfrak{C}) = |\mathrm{Interface}(\mathfrak{C})|$。

我们首先证明，对于每个 $\mathfrak{C} \in \mathrm{Coll}(T_{\mathrm{pref}}, X)$，确实有 $\widetilde{\mathrm{collmap}}(\mathfrak{C}) \in F \cup \mathrm{App}(T_{\mathrm{pref}})$。首先，$\mathfrak{C}$ 未阻塞的情形由引理 4.21立即得到解决：我们总有$\widetilde{\mathrm{collmap}}(\mathfrak{C}) \in F$。另一方面，如果 $\mathfrak{C}$ 受阻，则 t 是一个阻塞，而不是 T 的根，因为$T_{\mathrm{pref}} \neq \emptyset$。因此，$\mathrm{parent}(t)$存在且属于 F。如果 $\mathrm{parent}(t) \in T_{\mathrm{pref}}$，则 $\mathrm{collmap}(\mathfrak{C}) = t \in \mathrm{App}(T_{\mathrm{pref}})$。否则，$\mathrm{collmap}(\mathfrak{C}) = \mathrm{parent}(t) \in F$。

性质 (i) 显然成立。接下来证明性质 (ii)。考虑 $\mathfrak{C} \in \mathrm{Coll}(T_{\mathrm{pref}}, X)$ with $\mathrm{collmap}(\mathfrak{C}) \notin \mathrm{App}(T_{\mathrm{pref}})$。同样，未阻塞的收集分量的情形可由引理 4.21 解决。现在假设 $\mathfrak{C}$ 受阻。由于 $\mathrm{collmap}(\mathfrak{C}) \notin \mathrm{App}(T_{\mathrm{pref}})$，通过考察 collmap 的定义可知，$\mathrm{parent}(t) \notin T_{\mathrm{pref}}$ and $\mathrm{collmap}(\mathfrak{C}) = \mathrm{parent}(t)$。但此时可应用引理 4.22，得到 $\mathrm{weight}(\mathfrak{C}) = |\mathrm{bag}(t) \cap X| < |\mathrm{bag}(\mathrm{parent}(t))| = |\mathrm{bag}(\mathrm{collmap}(\mathfrak{C}))|$。

接下来需要证明性质(iii)，考虑一个节点 $t \notin T_{\mathrm{pref}}$。我们有 $|\mathrm{bag}(t)| \leqslant \ell + 1$。注意，每个未受阻连通分量 $\mathfrak{C}$ 都由 collmap 分配给某个已探索节点 t'，使得 $\mathrm{bag}(t')$ 与 $\mathfrak{C}$相交。由于收集分量构成了$V(G) \setminus X$的一个划分，至多有 $\ell + 1$ 个不同的收集分量与 $\mathrm{bag}(t)$ 相交，因此 collmap 至多能将$\ell + 1$ 个不同的未阻塞收集分量分配 t。接下来，每个受阻的收集分量 $\mathfrak{C}$ 都会由 ξ映射到不同的阻塞集 $b \in \mathrm{Blockages}(T_{\mathrm{pref}}, X)$。注意，根据 collmap 的定义，只有当 b 等于 t 或 t的一个子节点时，$\mathfrak{C}$ 才可能被 collmap 映射到 t。由于 T 是二叉的，我们得出结论：collmap 至多能将三个不同的受阻连通分量映射到 t。

□

现在，我们证明性质 (POT)的一个略强的版本。注意，由于 ℓ 是固定的，在本节余下部分中，我们省略势函数下标中的它；也就是说，我们使用 $\Phi := \Phi_\ell$。

引理 5.23。 以下不等式成立：

$$\Phi(\mathcal{T}) - \Phi(\mathcal{T}') \geqslant |\widetilde{F}| + \sum_{t \in T_{\mathrm{pref}}} \mathsf{height}_T(t) - k^{\mathcal{O}(k)} \cdot \left(|T_{\mathrm{pref}}| + \sum_{t \in \mathtt{App}(T_{\mathrm{pref}})} \mathsf{height}_T(t) \right) \cdot \log N.$$

证明。 为使证明更清晰，我们用 $\Phi_{\mathcal{T}}(\cdot)$ 表示在 $\mathcal{T}$ 中计算的 $\mathcal{T}$ 的节点上的势函数，用 $\Phi_{\mathcal{T}'}(\cdot)$ 表示在 $\mathcal{T}'$ 中计算的 $\mathcal{T}'$ 的节点上的势函数。此外，对于 T 的一个子树 S，记 $\Phi_{\mathcal{T}}(S) := \Phi_{\mathcal{T}}(V(S)) = \sum_{t \in V(S)} \Phi_{\mathcal{T}}(t)$，对于 $\Phi_{\mathcal{T}'}(\cdot)$ 也作类似定义。

对于树分解 $\mathcal{T}$，我们考虑其节点的如下划分：前缀 T_{pref}，$\widetilde{\text{除}}$ T_{pref} 之外所有已探索顶点的集合 F，以及未探索顶点的集合。因此，我们可以将 $\Phi(\mathcal{T})$ 写成以下形式：

$$\Phi(\mathcal{T}) = \Phi_{\mathcal{T}}(T_{\mathrm{pref}}) + \Phi_{\mathcal{T}}(\widetilde{F}) + \sum_{t \notin F} \Phi_{\mathcal{T}}(t).$$

对每个 $t \in T_{\mathrm{pref}}$ 应用不等式 $\Phi_{\mathcal{T}}(t) \geqslant$ 高度$_T(t)$，我们得到

$$\Phi(\mathcal{T}) \geqslant \sum_{t \in T_{\mathrm{pref}}} \mathsf{height}_T(t) + \Phi_{\mathcal{T}}(\widetilde{F}) + \sum_{t \notin F} \Phi_{\mathcal{T}}(t). \tag{3}$$

现在，我们来关注树分解 $\mathcal{T}'$ 的结构。根据 步骤 5 我们知道$V(T')$是 $V(T^X)$和 $V(T^B)$ 对每个 $B \in \mathcal{I}(T_{\mathrm{pref}},\ X)$ 的不相交并集。因此，

$$\Phi(\mathcal{T}') = \Phi_{\mathcal{T}'}(T^X) + \sum_{B \in \mathcal{I}(T_{\mathrm{pref}}, X)} \Phi_{\mathcal{T}'}(T^B).$$

由步骤 4 可知，每棵树 T^B都有一个前缀 $T^B{}_{\mathrm{pref}}$，并附有收集分量的分解$\mathcal{T}^{\mathfrak{C}}$，且满足：对于每个连通分量 $\mathfrak{C} \in \mathsf{Coll}(T_{\mathrm{pref}}, X)$，都存在唯一一棵树 T^B_{pref}，$T^{\mathfrak{C}}$ 附加于该树。因此，

$$\Phi(\mathcal{T}') = \Phi_{\mathcal{T}'}(T^X) + \sum_{B \in \mathcal{I}(T_{\mathrm{pref}}, X)} \Phi_{\mathcal{T}'}(T^B_{\mathrm{pref}}) + \sum_{\mathfrak{C} \in \mathsf{Coll}(T_{\mathrm{pref}}, X)} \Phi_{\mathcal{T}'}(T^{\mathfrak{C}}). \tag{4}$$

现在，我们将分别界定 式 (4) 右侧的三个项。

命题 5.24。 以下不等式成立：

$$\Phi_{\mathcal{T}'}(T^X) \leqslant k^{\mathcal{O}(k)} \cdot \left(|T_{\mathrm{pref}}| + \sum_{t \in \mathtt{App}(T_{\mathrm{pref}})} \mathsf{height}_T(t) \right) \cdot \log N.$$

命题证明。 为简洁起见，我们引入以下记号：对于两个节点 $t, t' \in V(T')$，若 t 是 T'中的 t' 的祖先，则记作 $t \preceq t'$。

回忆一下，$\mathcal{T}'$是通过将子树 T^B附加到 T^X的不同叶节点t_B上而构造的，其中每个子树对应一个 $B \in \mathcal{I}(T_{\mathrm{pref}}, X)$。因此，对于每个 $t \in V(T^X)$，有：

$$\mathsf{height}_{T'}(t) \leqslant \mathsf{height}(T^X) + \max\{\mathsf{height}(T^B) :\ B \in \mathcal{I}(T_{\mathrm{pref}}, X) \text{ and } t \preceq t_B\}. \tag{5}$$

（我们假设空集合的最大值为0。）回忆一下，高度（T^X）$\leqslant \mathcal{O}$（对数 $N+k$）（性质（P1））， 并且对于每个 $B \in \mathcal{I}$（T_{pref}，X），有

$$\mathsf{height}(T^B) \leqslant \mathcal{O}(\log N) + \max_{\mathfrak{C} \in \mathsf{Interface}^{-1}(B)} \mathsf{height}(T^{\mathfrak{C}})$$

（性质（P11））。现在，对于每个 $B \in \mathcal{I}$（T_{pref}，X），令我们固定 $\mathfrak{C}_B$为任意一个使$\mathfrak{C} \in$ 接口$^{-1}$（B）最大化高度（$T^{\mathfrak{C}}$）的收集分量。

将上述不等式代入公式（5），可得

$$\mathsf{height}_{T'}(t) \leqslant \mathcal{O}(\log N + k) + \max\{\mathsf{height}(T^{\mathfrak{C}_B}) :\ B \in \mathcal{I}(T_{\mathrm{pref}}, X) \text{ and } t \preceq t_B\}.$$

定义一个函数 δ：$V(T^X) \to \mathbb{N}$，以及一个部分函数 Λ：$V(T^X) \rightharpoonup \mathsf{Coll}(T_{\mathrm{pref}},\ X)$，如下所示：对于 $t \in V(T^X)$，令 $\Lambda(t)$ 为高度最大的收集分量 $\mathfrak{C}_B$，使得 $t \preceq t_B$（如果存在这样的分量）。随后，令 $\delta(t) := \mathsf{height}(\Lambda(t))$；或者，如果 $\Lambda(t)$ 未定义，则令 $\delta(t) := 0$。注意，$\mathsf{height}_{T'}(t) \leqslant \mathcal{O}(\log N + k) + \delta(t)$；因此，

$$\begin{aligned}
\Phi_{\mathcal{T}'}(T^X) &= \sum_{t \in V(T^X)} g(|\mathsf{bag}'(t)|) \cdot \mathsf{height}_{T'}(t) \leqslant \sum_{t \in V(T^X)} g(\ell+1) \cdot (\mathcal{O}(\log N + k) + \delta(t)) \\
&\leqslant \sum_{t \in V(T^X)} k^{\mathcal{O}(k)} \cdot (\log N + \delta(t)) \\
&\overset{(\star)}{\leqslant} k^{\mathcal{O}(k)} \cdot |T_{\mathrm{pref}}| \cdot \log N \ + \ k^{\mathcal{O}(k)} \cdot \sum_{t \in V(T^X)} \delta(t).
\end{aligned} \tag{6}$$

（在（$\star$）中，我们利用了以下事实：$|V(T^X)| \leqslant 2^{\mathcal{O}(k)} \cdot |T_{\mathrm{pref}}|$ 根据性质（P2）。）因此，剩下的任务是给 $\sum_{t \in V(T^X)} \delta(t)$ 一个界。

令 $B \in \mathcal{I}(T_{\mathrm{pref}}, X)$ 为一个接口。由于 T^X 是一棵高度为 $\mathcal{O}(\log N + k)$ 的树，t_B 在 T^X 中至多有 $\mathcal{O}(\log N + k)$ 个祖先。因此，至多有 $\mathcal{O}(\log N + k)$ 个节点 $t \in V(T^X)$ 满足 $\Lambda(t) = \mathfrak{C}_B$。显然，对于每个这样的 t，都有 $\delta(t) = \mathsf{height}(T^{\mathfrak{C}_B})$。因此，

$$\sum_{t \in V(T^X)} \delta(t) \leqslant \sum_{B \in \mathcal{I}(T_{\mathrm{pref}}, X)} \mathcal{O}(\log N + k) \cdot \mathsf{height}(T^{\mathfrak{C}_B}). \tag{7}$$

对于每个接口 B，令 $a_B \in \mathsf{App}(T_{\mathrm{pref}})$ 为 T_{pref} 中满足 $\mathfrak{C}_B \subseteq \mathsf{cmp}(a_B)$ 的唯一附属节点。根据引理 5.14，对于每个附属节点 $t \in \mathsf{App}(T_{\mathrm{pref}})$，至多有 $k^{\mathcal{O}(k)}$ 个接口 B 满足 $a_B = t$。对于每个这样的接口 B，$\mathsf{height}(T^{\mathfrak{C}_B})$ 至多为 T 中 $\mathfrak{C}_B$ 的归属袋的高度（性质 (P6)）。由于 $\mathfrak{C}_B \subseteq \mathsf{cmp}(t)$，$\mathfrak{C}_B$ 的归属袋是 t 的后代。因此，$\mathsf{height}(T^{\mathfrak{C}_B}) \leqslant \mathsf{height}_T(t)$。结合式 (7)，可得

$$\sum_{t \in V(T^X)} \delta(t) \leqslant \mathcal{O}(\log N + k) \cdot \sum_{t \in \mathsf{App}(T_{\mathrm{pref}})} k^{\mathcal{O}(k)} \cdot \mathsf{height}_T(t) \leqslant k^{\mathcal{O}(k)} \cdot \sum_{t \in \mathsf{App}(T_{\mathrm{pref}})} \mathsf{height}_T(t) \cdot \log N.$$

结合 式 (6)，证明至此完成。 ◁

断言 5.25。 以下不等式成立：

$$\sum_{B \in \mathcal{I}(T_{\mathrm{pref}}, X)} \Phi_{\mathcal{T}'}(T^B_{\mathrm{pref}}) \leqslant 52(\ell+4) \cdot \left(\sum_{t \in \widetilde{F}} g(|\mathsf{bag}(t)| - 1) \cdot \mathsf{height}_T(t) \right) + \sum_{t \in \mathsf{App}(T_{\mathrm{pref}})} k^{\mathcal{O}(k)} \cdot \mathsf{height}_T(t).$$

命题证明。 固定一个集合 $B \in \mathcal{I}(T_{\mathrm{pref}}, X)$。回顾一下，

$$\sum_{t \in V(T^B_{\mathrm{pref}})} \mathsf{height}_{T^B}(t) \leqslant 52 \cdot \sum_{\mathfrak{C} \in \mathsf{Interface}^{-1}(B)} \mathsf{height}(T^{\mathfrak{C}})$$

（性质(P13)）。因此，

$$\begin{aligned}
\Phi_{\mathcal{T}'}(T^B_{\mathrm{pref}}) &= \sum_{t \in T^B_{\mathrm{pref}}} g(|B|) \cdot \mathsf{height}_{T'}(t) = \sum_{t \in T^B_{\mathrm{pref}}} g(|B|) \cdot \mathsf{height}_{T^B}(t) \\
&\leqslant g(|B|) \cdot 52 \cdot \sum_{\mathfrak{C} \in \mathsf{Interface}^{-1}(B)} \mathsf{height}(T^{\mathfrak{C}}) \\
&\overset{(\star)}{\leqslant} 52 \cdot \sum_{\mathfrak{C} \in \mathsf{Interface}^{-1}(B)} g(\mathsf{weight}(\mathfrak{C})) \cdot \mathsf{height}(\mathfrak{C}).
\end{aligned}$$

注意到，$(\star)$源于以下事实：对于 $\mathfrak{C} \in \mathsf{Interface}^{-1}(B)$（定义 8），$\mathsf{weight}(\mathfrak{C}) = |\mathsf{Interface}(\mathfrak{C})| = |B|$；并且 $\mathsf{height}_T(\mathfrak{C})$，或者根据定义，$\mathfrak{C}$ 的归属袋的高度，至少为 $\mathsf{height}(T^{\mathfrak{C}})$（性质（P6））。因此，将上述内容对所有集合 $B \in \mathcal{I}(T_{\mathrm{pref}}, X)$ 求和后，我们得到

$$\sum_{B \in \mathcal{I}(T_{\mathrm{pref}}, X)} \Phi_{\mathcal{T}'}(T^B_{\mathrm{pref}}) \leqslant 52 \cdot \sum_{\mathfrak{C} \in \mathsf{Coll}(T_{\mathrm{pref}}, X)} g(\mathsf{weight}(\mathfrak{C})) \cdot \mathsf{height}(\mathfrak{C}).$$

~现在我们使用引理 5.22。令$\mathsf{collmap} : \mathsf{Coll}(T_{\mathrm{pref}},\ X) \to F \cup \mathtt{App}(T_{\mathrm{pref}})$ 为该引理中的映射。我们将上面的和拆分为两项：

$$\sum_{\mathfrak{C}\in\mathsf{Coll}(T_{\mathrm{pref}},X)} g(\mathsf{weight}(\mathfrak{C}))\cdot\mathsf{height}(\mathfrak{C}) \leqslant \sum_{\substack{\mathfrak{C}\in\mathsf{Coll}(T_{\mathrm{pref}},X)\\ \mathsf{collmap}(\mathfrak{C})\in\widetilde{F}}} g(\mathsf{weight}(\mathfrak{C}))\cdot\mathsf{height}(\mathfrak{C}) + \sum_{\substack{\mathfrak{C}\in\mathsf{Coll}(T_{\mathrm{pref}},X)\\ \mathsf{collmap}(\mathfrak{C})\in\mathtt{App}(T_{\mathrm{pref}})}} g(\mathsf{weight}(\mathfrak{C}))\cdot\mathsf{height}(\mathfrak{C}). \tag{8}$$

~（该不等式源于一种可能性：$\mathtt{App}(T_{\mathrm{pref}}) \cap F \neq \emptyset$。）为了界定右侧的第一个~求和项，我们利用以下事实：若 $\mathsf{collmap}(\mathfrak{C}) \in F$，则

- $\mathsf{权重}(\mathfrak{C}) < |\mathsf{袋}(\mathsf{collmap}(\mathfrak{C}))|$，
- $\mathsf{height}(\mathfrak{C}) \leqslant \mathsf{height}_T(\mathsf{collmap}(\mathfrak{C}))$，并且
- $|\mathsf{collmap}^{-1}(t)| \leqslant \ell + 4$，对每个 $t \in \widetilde{F}$ 均成立。

因此，

$$\sum_{\substack{\mathfrak{C}\in\mathsf{Coll}(T_{\mathrm{pref}},X)\\ \mathsf{collmap}(\mathfrak{C})\in\widetilde{F}}} g(\mathsf{weight}(\mathfrak{C}))\cdot\mathsf{height}(\mathfrak{C}) \leqslant \sum_{\substack{\mathfrak{C}\in\mathsf{Coll}(T_{\mathrm{pref}},X)\\ \mathsf{collmap}(\mathfrak{C})\in\widetilde{F}}} g(|\mathsf{bag}(\mathsf{collmap}(\mathfrak{C}))|-1)\cdot\mathsf{height}_T(\mathsf{collmap}(\mathfrak{C})) \leqslant \sum_{t\in\widetilde{F}} (\ell+4)\cdot g(|\mathsf{bag}(t)|-1)\cdot\mathsf{height}_T(t). \tag{9}$$

为了界定右侧的第二个求和项，我们利用式 (8)中的事实：若$\mathsf{collmap}(\mathfrak{C}) \in \mathtt{App}(T_{\mathrm{pref}})$，则

- $\mathsf{权重}(\mathfrak{C}) = |\mathsf{接口}(\mathfrak{C})| \leqslant \ell + 1 = 6k + 6$（这是因为$\mathsf{接口}(\mathfrak{C})$ 是$\mathsf{接口}(\mathfrak{C})$ 的归属袋的子集，这由性质(P5)可知），
- $\mathsf{高度}(\mathfrak{C}) \leqslant \mathsf{高度}_T(\mathsf{collmap}(\mathfrak{C}))$，以及
- $|\mathsf{collmap}^{-1}(t)| \leqslant \ell + 4$，对于每个 $t \in \mathtt{App}(T_{\mathrm{pref}})$。

因此，

$$\sum_{\substack{\mathfrak{C}\in\mathsf{Coll}(T_{\mathrm{pref}},X)\\ \mathsf{collmap}(\mathfrak{C})\in\mathtt{App}(T_{\mathrm{pref}})}} g(\mathsf{weight}(\mathfrak{C}))\cdot\mathsf{height}(\mathfrak{C}) \leqslant \sum_{\substack{\mathfrak{C}\in\mathsf{Coll}(T_{\mathrm{pref}},X)\\ \mathsf{collmap}(\mathfrak{C})\in\mathtt{App}(T_{\mathrm{pref}})}} g(6k+6)\cdot\mathsf{height}_T(\mathsf{collmap}(\mathfrak{C})) \leqslant \sum_{t\in\mathtt{App}(T_{\mathrm{pref}})} (\ell+4)\cdot g(6k+6)\cdot\mathsf{height}_T(t) \leqslant \sum_{t\in\mathtt{App}(T_{\mathrm{pref}})} k^{\mathcal{O}(k)}\cdot\mathsf{height}_T(t). \tag{10}$$

By 代入公式（9）和公式（10）到不等式公式（8）中，可得所需的界 d. ◁

命题 5.26. 以下不等式成立：

$$\sum_{\mathfrak{C}\in\mathsf{Coll}(T_{\mathrm{pref}},X)} \Phi_{\mathcal{T}'}(T^{\mathfrak{C}}) \leqslant (\ell+1)\cdot\left(\sum_{t\in\widetilde{F}} g(|\mathsf{bag}(t)|-1)\cdot \mathsf{height}_T(t)\right) + \sum_{t\notin F}\Phi_{\mathcal{T}}(t).$$

命题证明。我们有

$$\sum_{\mathfrak{C}\in\mathsf{Coll}(T_{\mathrm{pref}},X)} \Phi_{\mathcal{T}'}(T^{\mathfrak{C}}) = \sum_{\mathfrak{C}\in\mathsf{Coll}(T_{\mathrm{pref}},X)}\ \sum_{t\in V(T^{\mathfrak{C}})} \Phi_{\mathcal{T}'}(t)$$

对于每个节点 $t\in V(T^{\mathfrak{C}})$，考虑它在 T 中的原始副本 $\mathsf{origin}(t)$。注意，$\widetilde{\ }\mathsf{origin}(t)$ 要么是一个已探索节点，此时 $\mathsf{origin}(t)\in F$；要么是阻塞的后代，此时 $\mathsf{origin}(t)\notin F$。因此：

$$\sum_{\mathfrak{C}\in\mathsf{Coll}(T_{\mathrm{pref}},X)} \Phi_{\mathcal{T}'}(T^{\mathfrak{C}}) = \sum_{\mathfrak{C}\in\mathsf{Coll}(T_{\mathrm{pref}},X)}\ \sum_{\substack{t\in V(T^{\mathfrak{C}})\\ \mathsf{origin}(t)\in\widetilde{F}}} \Phi_{\mathcal{T}'}(t) + \sum_{\mathfrak{C}\in\mathsf{Coll}(T_{\mathrm{pref}},X)}\ \sum_{\substack{t\in V(T^{\mathfrak{C}})\\ \mathsf{origin}(t)\notin F}} \Phi_{\mathcal{T}'}(t). \tag{11}$$

回顾一下，对于每个 $t\in V(T^{\mathfrak{C}})$，如果 $\mathsf{origin}(t)$ 未被探索，那么 $|\text{袋}^{\mathfrak{C}}(t)| = |\text{袋}(\mathsf{origin}(t))|$ 且$|\text{原节点}^{-1}(\mathsf{origin}(t))| = 1$（性质 (P8)），并且$\text{高度}_{T^{\mathfrak{C}}}(t) \leqslant \text{高度}_T(\mathsf{origin}(t))$（性质 (P9)）。由于对于每个 $t\in V(T^{\mathfrak{C}})$，$\text{高度}_{T'}(t) = \text{高度}_{T^{\mathfrak{C}}}(t)$，我们得到

$$\begin{aligned}
\sum_{\mathfrak{C}\in\mathsf{Coll}(T_{\mathrm{pref}},X)}\ \sum_{\substack{t\in V(T^{\mathfrak{C}})\\ \mathsf{origin}(t)\notin F}} \Phi_{\mathcal{T}'}(t) &= \sum_{\mathfrak{C}\in\mathsf{Coll}(T_{\mathrm{pref}},X)}\ \sum_{\substack{t\in V(T^{\mathfrak{C}})\\ \mathsf{origin}(t)\notin F}} g(|\mathsf{bag}^{\mathfrak{C}}(t)|)\cdot \mathsf{height}_{T'}(t)\\
&\leqslant \sum_{\mathfrak{C}\in\mathsf{Coll}(T_{\mathrm{pref}},X)}\ \sum_{\substack{t\in V(T^{\mathfrak{C}})\\ \mathsf{origin}(t)\notin F}} g(|\mathsf{bag}(\mathsf{origin}(t))|)\cdot \mathsf{height}_T(\mathsf{origin}(t))\\
&\leqslant \sum_{t\notin F} g(|\mathsf{bag}(t)|)\cdot \mathsf{height}_T(t) = \sum_{t\notin F}\Phi_{\mathcal{T}}(t).
\end{aligned} \tag{12}$$

注意，上述情况中可能存在这样的未探索节点 $t\in V(T)\setminus F$，使得$\text{原节点}^{-1}(t) = \emptyset$。

现在，考虑已探索节点。根据以下事实：对于每个带有已探索原节点 t 的 $t\in V(T^{\mathfrak{C}})$，我们有 $|\text{袋}^{\mathfrak{C}}(t)| < |\text{袋}(\text{原节点}(t))|$，并且 $|\text{原节点}^{-1}(\text{原节点}(t))| \leqslant \ell+1$（性质 (P7)）；此外，$T^{\mathfrak{C}}(t)$ 的高度 $\leqslant \text{高度}_T(\text{原节点}(t))$（同样根据性质 (P9)），我们得到

$$\begin{aligned}
\sum_{\mathfrak{C}\in\mathsf{Coll}(T_{\mathrm{pref}},X)}\ \sum_{\substack{t\in V(T^{\mathfrak{C}})\\ \mathsf{origin}(t)\in\widetilde{F}}} \Phi_{\mathcal{T}'}(t) &= \sum_{\mathfrak{C}\in\mathsf{Coll}(T_{\mathrm{pref}},X)}\ \sum_{\substack{t\in V(T^{\mathfrak{C}})\\ \mathsf{origin}(t)\in\widetilde{F}}} g(|\mathsf{bag}^{\mathfrak{C}}(t)|)\cdot \mathsf{height}_{T'}(t)\\
&\leqslant \sum_{\mathfrak{C}\in\mathsf{Coll}(T_{\mathrm{pref}},X)}\ \sum_{\substack{t\in V(T^{\mathfrak{C}})\\ \mathsf{origin}(t)\in\widetilde{F}}} g(|\mathsf{bag}(\mathsf{origin}(t))|-1)\cdot \mathsf{height}_T(\mathsf{origin}(t))\\
&\leqslant \sum_{t\in\widetilde{F}} (\ell+1)\cdot g(|\mathsf{bag}(t)-1|)\cdot \mathsf{height}_T(t).
\end{aligned}$$

将上述界与式（12）代入式（11），即可得到所需的不等式。◁

然后结合上述结果，得到 $\mathcal{T}'$的势函数界。

命题 5.27。 以下不等式成立：

$$\Phi(\mathcal{T}') \leqslant \Phi_{\mathcal{T}}(\widetilde{F}) - |\widetilde{F}| + \sum_{t \notin F} \Phi_{\mathcal{T}}(t) + k^{\mathcal{O}(k)} \cdot \left(|T_{\mathrm{pref}}| + \sum_{t \in \mathtt{App}(T_{\mathrm{pref}})} \mathsf{height}_T(t) \right) \cdot \log N$$

命题证明。 回顾一下，公式（4）：

$$\Phi(\mathcal{T}') = \Phi_{\mathcal{T}'}(T^X) + \sum_{B \in \mathcal{I}(T_{\mathrm{pref}}, X)} \Phi_{\mathcal{T}'}(T^B_{\mathrm{pref}}) + \sum_{\mathfrak{C} \in \mathsf{Coll}(T_{\mathrm{pref}}, X)} \Phi_{\mathcal{T}'}(T^{\mathfrak{C}}).$$

应用命题 5.24、断言 5.25 和命题 5.26中的不等式，可得：

$$\begin{aligned}
\Phi(\mathcal{T}') \leqslant\ & k^{\mathcal{O}(k)} \cdot \left(|T_{\mathrm{pref}}| + \sum_{t \in \mathtt{App}(T_{\mathrm{pref}})} \mathsf{height}_T(t) \right) \cdot \log N \\
& + 52(\ell+4) \cdot \left(\sum_{t \in \widetilde{F}} g(|\mathsf{bag}(t)| - 1) \cdot \mathsf{height}_T(t) \right) + \sum_{t \in \mathtt{App}(T_{\mathrm{pref}})} k^{\mathcal{O}(k)} \cdot \mathsf{height}_T(t) \\
& + (\ell+1) \cdot \left(\sum_{t \in \widetilde{F}} g(|\mathsf{bag}(t)| - 1) \cdot \mathsf{height}_T(t) \right) + \sum_{t \notin F} \Phi_{\mathcal{T}}(t).
\end{aligned}$$

检查 $g(\cdot)$ 在第 5.1 节 中的定义可知

$$g(x) \geqslant (53\ell + 209) \cdot g(x-1) + 1 \qquad \text{for every } x \in \mathbb{Z}_{\geqslant 1}.$$

利用这一点，我们可以对上述不等式中的两项进行界定：

$$\begin{aligned}
& 52(\ell+4) \cdot \left(\sum_{t \in \widetilde{F}} g(|\mathsf{bag}(t)| - 1) \cdot \mathsf{height}_T(t) \right) + (\ell+1) \cdot \left(\sum_{t \in \widetilde{F}} g(|\mathsf{bag}(t)| - 1) \cdot \mathsf{height}_T(t) \right) \\
=\ & \sum_{t \in \widetilde{F}} (53\ell + 209) \cdot g(|\mathsf{bag}(t)| - 1) \cdot \mathsf{height}_T(t) \\
\leqslant\ & \sum_{t \in \widetilde{F}} (g(|\mathsf{bag}(t)|) \cdot \mathsf{height}_T(t) - 1) = \Phi_{\mathcal{T}}(\widetilde{F}) - |\widetilde{F}|.
\end{aligned}$$

因此，

$$\Phi(\mathcal{T}') \leqslant \Phi_{\mathcal{T}}(\widetilde{F}) - |\widetilde{F}| + \sum_{t \notin F} \Phi_{\mathcal{T}}(t) + k^{\mathcal{O}(k)} \cdot \left(|T_{\mathrm{pref}}| + \sum_{t \in \mathtt{App}(T_{\mathrm{pref}})} \mathsf{height}_T(t) \right) \cdot \log N,$$

如所期望。 ◁

我们现在可以证明引理 5.23。回顾公式（3）：

$$\Phi(\mathcal{T}) \geqslant \sum_{t \in T_{\mathrm{pref}}} \mathsf{height}_T(t) + \Phi_{\mathcal{T}}(\widetilde{F}) + \sum_{t \notin F} \Phi_{\mathcal{T}}(t).$$

将该不等式与命题 5.27给出的界结合起来，可得：

$$\Phi(\mathcal{T}) - \Phi(\mathcal{T}') \geqslant \sum_{t \in T_{\mathrm{pref}}} \mathsf{height}_T(t) + |\widetilde{F}| - k^{\mathcal{O}(k)} \cdot \left(|T_{\mathrm{pref}}| + \sum_{t \in \mathtt{App}(T_{\mathrm{pref}})} \mathsf{height}_T(t) \right) \cdot \log N;$$

至此，引理的证明结束。 □

显然，引理 5.23 蕴含细化操作的性质 (POT) 。此外，我们可以据此~推导出 F的规模界：

$$|\widetilde{F}| \leqslant \Phi(\mathcal{T}) - \Phi(\mathcal{T}') - \sum_{t \in T_{\mathrm{pref}}} \mathsf{height}_T(t) + k^{\mathcal{O}(k)} \cdot \left(|T_{\mathrm{pref}}| + \sum_{t \in \mathtt{App}(T_{\mathrm{pref}})} \mathsf{height}_T(t) \right) \cdot \log N.$$

回想一下，根据事实 5.21，细化调用的运行时间上界为 ：

$$2^{\mathcal{O}(k^9)}(|\widetilde{F}| + |T_{\mathrm{pref}}| \cdot \log N)$$

~利用得到的 $|F|$界，我们便可得到性质(RT1)所给出的运行时间期望界。

6 高度改进

本节利用第 5 节 中定义的细化操作，构造一种能够维护小高度树分解的数据结构。与第 5 节一样，我们假设 $\ell = 6k+5$并取 $\Phi := \Phi_\ell$ 和 $g := g_\ell$，其定义见第 5.1 节。我们证明：

引理 6.1 （高度改进数据结构）。 固定 $k \in \mathbb{N}$ 并令 $\ell = 6k + 5$。引理 $(\ell + 1)$- 前缀重建数据结构（来自 引理 *5.1* ，用于维护树分解 $\mathcal{T} = (T,$ 袋$)$ 可以扩展为额外支持以下操作：

- improveHeight()：通过一系列前缀重建更新来更新 $\mathcal{T}$，生成一个树分解 $\mathcal{T}' = (T',$ 袋′$)$，使得

$$\mathsf{height}(T') \leqslant 2^{\mathcal{O}(k \log k \sqrt{\log n \log\log n})} \quad \textit{and} \quad |V(T')| \leqslant k^{\mathcal{O}(k)} \cdot n^3.$$

此外，$\Phi(\mathcal{T}') \leqslant \Phi(\mathcal{T})$，并且如果 $\mathcal{T}$ 的宽度至多为 ℓ，那么 $\mathcal{T}'$ 的宽度也至多为 ℓ。improveHeight 的最坏情况运行时间有如下界：

$$2^{\mathcal{O}(k^9)} \cdot (\Phi(\mathcal{T}) - \Phi(\mathcal{T}')) + \mathcal{O}(1).$$

引理 6.1对于确保动态维护图的树分解的数据结构的效率至关重要：每次更新树分解后，我们都会调用improveHeight ，以确保分解的高度足够小。需要注意的是，为降低分解高度而由improveHeight 执行的所有前缀重建更新，就摊还运行时间而言基本上都是“免费的”：改进的运行时间完全由势值的下降来摊还，即 $\Phi(\mathcal{T}) - \Phi(\mathcal{T}')$。特别地，势值的下降始终非负。

本节其余部分将用于证明引理 6.1。

6.1 算法描述

我们首先介绍 improveHeight。假设给定一个树分解 $(T, \text{袋})$，并且可以访问引理 3.5 中的 $(\ell+1)$-前缀重建数据结构；特别地，我们可以在常数时间内访问每个节点 s 的高度 height(s) 和规模 size(s)，即以 s 为根的 T 的子树中的节点数。树分解只会通过引理 5.1 实现的细化操作 refine($\cdot$)，或通过从头重新计算 $\mathcal{T}$ 来进行更新。

首先，我们定义一个递归函数 getUnbalanced 它假定树 T 是"不平-衡"的，并确定一个不平衡前缀 W 属于 T，随后我们会将其传递给 引理 5.1中的细化操作。树是否平衡取决于其高度（即树是深还是浅）和规模（即树是大还是小）：一棵树同时满足深且小，就是不平衡的。请注意，"深"和"小"这两个词的定义取决于当前递归的深度。

现在，我们将 getUnbalanced 的描述形式化。假设给定两个正整数序列：$h_1 > h_2 > \ldots > h_a$和 $n_1 > n_2 > \ldots > n_a$，其中 $h_j \leqslant n_j$对于$j < a$且 $h_a > n_a$。对于 T和 $j \in \{1, \ldots, a\}$中的节点 t，若以 t 为根的 T的子树 T_t满足高度(t) $< h_j$，则称其为 j-浅；否则称为 j-深。类似地，若该子树满足规模(t) $> n_j$，则称其为 j-大；否则称为 j-小（不要与 c-小闭包的概念混淆）。我们定义递归函数 getUnbalanced(T_t, j)，其中 T_t 是 T中以 $t \in V(T)$ 为根的子树，且 $j \in [a-1]$，定义如下。我们首先断言 T_t 既是 j-小的，也是 j-深的。令 P为从 t 到 T_t的一个最长路径，其中 T_t是 t 的一片叶节点。定义

$$\mathsf{UBApp}(P, j) := \{t' \in \mathsf{App}(P) : T_{t'} \text{ is } (j+1)\text{-deep and } (j+1)\text{-small}\}$$

作为以 P 的附属节点为根的不平衡树集合。另一方面，我们称以 App(P) $\setminus$ UBApp(P, j) 中某个节点为根的每棵树（即以 P 的附属节点为根的每棵平衡树）都被 P-找到。那么，getUnbalanced(T_t, j) 返回

$$P \cup \bigcup_{t' \in \mathsf{UBApp}(P,j)} \mathsf{getUnbalanced}(T_{t'}, j+1).$$

显然，getUnbalanced 定义良好：在每次递归调用 getUnbalanced($T_{t'}$, $j+1$) 时，都保证 $T_{t'}$ 同时是 $(j+1)$-深且 $(j+1)$-小；因此，由于 $h_a > n_a$，并且每棵树的高度都不超过其规模，getUnbalanced($\cdot$, a) 不会被调用。此外，很容易验证 getUnbalanced(T, 1) 返回 T 的一个前缀。

介绍完 getUnbalanced 后，我们说明如何在 improveHeight 过程的实现中使用它。我们假设该数据结构可以访问引理 5.1 中的细化前缀重建数据结构。首先，确保 T 的规模合理：如果 $|V(T)|$ 至少为 $B \cdot n^3$，其中 $B := g_\ell(\ell+1) \leqslant k^{\mathcal{O}(k)}$，那么我们使用定理 5.3 中的算法，在 $\mathcal{T}$ 上从头重新计算整个树分解，并将其转换为二叉树，同时使其规模增加不超过两倍，从而得到一个新的树分解 $\mathcal{T}^\star = (T^\star, \text{袋}^\star)$。我们将此过程称为缩减。如果发生缩减，就丢弃旧的细化数据结构，并用 $\mathcal{T}^\star$ 初始化一个新的数据结构（注意，这可以表示为一次规模为 $|V(T)| + |V(T^\star)|$ 的前缀重建更新）。注意，$\mathcal{T}^\star$ 是一个宽度至多为 k、规模至多为 $\mathcal{O}(n)$ 的树分解，因此 $\Phi(\mathcal{T}^\star) = \mathcal{O}(B \cdot n^2)$。注意，由于 $\Phi(\mathcal{T}) \geqslant B \cdot n^3 \gg \Phi(\mathcal{T}^\star)$，势值的减少将抵消缩减所产生的计算开销。

确保 T 的规模合理后，我们检查 T 是否为 1-深。如果不是，我们就完成了。如果是，我们就通过 getUnbalanced(T,1) 提取一个不平衡前缀 W，我们称之为

refine(W)，并将其返回的前缀重建更新应用于 (T, 袋)。参数 n_1 将选择为使得 $n_1 \geqslant B \cdot n^3 \geqslant |V(T)|$。因此，在调用 getUnbalanced($T$, 1) 时， T 是 1-深且 1-小的，所以满足 getUnbalanced 的初始不变量。我们稍后将证明，对于适当选择的序列 (n_i)、(h_i)，势值将足以覆盖细化的时间复杂度。

现在，经过一次该过程后， T 可能仍不够浅。因此，我们在循环中重复上述操作序列，直到树达到这样的状态；需要强调的是，在循环的每次迭代中，我们还会检查 T 是否过大，并在必要时将其缩小。由于每次循环迭代都会使势函数 $\Phi(\mathcal{T})$显著下降，因此这一过程必定有限；在下文中，我们也将能够合理地界定其运行时间。

在说明 getUnbalanced 为何满足 引理 6.1 中的界，以及为何它最终会终止之前，我们先定义序列 (h_i)和 (n_i)。

引理 6.2。 设 N 和 c 为满足 $1 < c < N$的实数。存在实数序列$h_1 > h_2 > \ldots > h_a$和 $n_1 > n_2 > \ldots > n_a$，使得

- $h_1 \leqslant 2^{\mathcal{O}(\sqrt{\log N \log c})}$且 $n_1 \geqslant N$，
- $h_a > n_a > 1$，并且

对所有 $1 \leqslant i < a$，都有 • $n_i = n_{i+1} \cdot h_{i+1} \geqslant c \cdot h_{i+1} = h_i$。

证明。 令 a 为满足 $c^{a_{(a+1)2}} \geqslant N$ 的最小整数。 我们令

$$h_i := c^{a+2-i} \qquad \text{and} \qquad n_i := c^{\frac{(a-i+1)\cdot(a-i+2)}{2}}.$$

对于 $1 \leqslant i < a$，我们有 $h_a = c^2 > c = n_a$ 和 $n_i = n_{i+1} \cdot h_{i+1} \geqslant c \cdot h_{i+1} = h_i$。此外，注意到$n_1 = c^{a_{(a+1)2}} \geqslant N$。因此，只需证明 $h_1 \leqslant 2^{\mathcal{O}(\sqrt{\log N \log c})}$。

由于 $c < N$，我们有 $a \geqslant 2$。根据 a 的定义，我们还有 $c^{a_{(a-1)^2}} < N \leqslant c^{a_{(a+1)2}}$，这意味着 $^{a}_{(a-1)2} \cdot \log c < \log N \leqslant {}^{a}_{(a+1)2} \cdot \log c$。由于 $a^2 \geqslant {}^{a}_{(a+1)2}$，我们还有$a^2 \log c \geqslant \log N$，因此 $a \geqslant \sqrt{\frac{\log N}{c \log}}$。另一方面，由于 $a \geqslant 2$，我们有 $a^2 \leqslant 4 \cdot {}^{a}_{(a-1)2}$，所以

$$a \log c = \frac{a^2 \log c}{a} \leqslant \frac{4^{\frac{a(a-1)}{2}} \log c}{a} \leqslant \frac{4 \log N}{a} \leqslant \frac{4 \log N}{\sqrt{\frac{\log N}{\log c}}} = 4\sqrt{\log N \log c}.$$

因此， $h_1 = c^{a+1} \leqslant c^{2a} = 2^{2a \log c} = 2^{\mathcal{O}(\sqrt{\log N \log c})}$。 □

通过应用引理 6.2 ，结合 $N = |V(T)|$和 $c = 2^{\mathcal{O}(k \log k)} \log n$，我们得到序列 ($h_i$) 和 ($n_i$)，具体内容稍后说明。

6.2 势函数下降的界

设 $\mathcal{T}'$ 为高度改进过程的一次循环迭代后得到的树分解，并假设未发生收缩过程。现在，我们将利用 $\Phi(\mathcal{T}) - \Phi(\mathcal{T}')$的性质 (POT) ，给出其界，该性质来自引理 5.1。

令 W 为 getUnbalanced(T, 1) 返回的 T 前缀，并令 $\mathcal{W}$ 为以 App(W) 中各顶点为根的 T 子树集合。由于 W 是由 getUnbalanced 找到的垂直路径的不相交并集，

因此可以为 $\mathcal{W}$ 中的每棵子树指定由 getUnbalanced 找到的唯一垂直路径 P，使得该子树是 P-found。

我们先关注递归调用 getUnbalanced(T_r, j)，其中 T_r 是以r 为根的 T的子树，且 $j \in [a-1]$。令 P为算法中找到的以 r为根的最长路径。令 T_1, T_2, ..., T_b为通过 P找到的子树集合。由于 T_r是二叉的，我们有 $b \leqslant |P|+1 \leqslant 2|P|$。根据 getUnbalanced 的不变量可知，T_r 是 j-深的，即 $|P| = \mathsf{height}(T_r) \geqslant h_j$。由于每个通过 P找到的子树要么是 $(j+1)$-浅的，要么是 $(j+1)$-大型的，因此我们有

$$\sum_{i=1}^{b} \mathsf{height}(T_i) \leqslant \sum_{i\,:\,T_i \text{ is } (j+1)\text{-shallow}} \mathsf{height}(T_i) + \sum_{i\,:\,T_i \text{ is } (j+1)\text{-big}} \mathsf{height}(T_i).$$

（注意，子树可以同时是 $(j+1)$-浅的和 $(j+1)$-大型的。）现在我们将分别界定右侧的每个求和项。

首先，每棵 $(j+1)$-浅子树的高度都小于 h_{j+1}。由于在通过 P找到的子树中，这类子树至多有 $b \leqslant 2|P|$棵，因此我们得到

$$\sum_{i\,:\,T_i \text{ is } (j+1)\text{-shallow}} \mathsf{height}(T_i) \leqslant 2|P| \cdot h_{j+1}.$$

其次， P是 T_r中最长的根到叶路径，因此每个 T_1, T_2, ..., T_b的高度至多为 $|P|$。此外，由于 T_r 是 j-小型，我们有 $|V(T_r)| \leqslant n_j$；对于每棵$(j+1)$-大型树 T_i ，我们有$|V(T_i)| > n_{j+1}$。由于 T_1, ..., T_b两两不相交，且都是 T_r的子树，因此我们得出， P找到的树中至多有$\frac{n_j}{n_{j+1}}$ 棵是$(j+1)$-大型的。因此，

$$\sum_{i\,:\,T_i \text{ is } (j+1)\text{-big}} \mathsf{height}(T_i) \leqslant 2|P| \cdot \frac{n_j}{n_{j+1}}.$$

我们得出，

$$\sum_{i=1}^{b} \mathsf{height}(T_i) \leqslant 2|P| \cdot \left(h_{j+1} + \frac{n_j}{n_{j+1}}\right). \tag{13}$$

此外，我们注意到，

$$\sum_{t \in P} \mathsf{height}(t) = 1 + 2 + \cdots + |P| \geqslant \frac{|P|^2}{2} \geqslant \frac{|P| \cdot h_j}{2}. \tag{14}$$

现在让我们回顾性质（POT），按照我们当前的记号调整后，它表述为

$$\Phi(\mathcal{T}) - \Phi(\mathcal{T}') \geqslant \sum_{t \in W} \mathsf{height}_T(t) - 2^{\mathcal{O}(k \log k)} \cdot \left(|W| + \sum_{t \in \mathtt{App}(W)} \mathsf{height}_T(t)\right) \cdot \log |V(T)|.$$

在该计算步骤中，我们有 $|V(T)| \leqslant B \cdot n^3$，其中 $B = k^{\mathcal{O}(k)}$，因此 $\log |V(T)| \leqslant \mathcal{O}(\log n + k \log k)$。因此，我们可以将这个界简化为：

$$\Phi(\mathcal{T}) - \Phi(\mathcal{T}') \geqslant \sum_{t \in W} \mathsf{height}_T(t) - 2^{\mathcal{O}(k \log k)} \cdot \left(|W| + \sum_{t \in \mathtt{App}(W)} \mathsf{height}_T(t)\right) \cdot \log n. \tag{15}$$

现在我们展开 $\mathcal{O}(\cdot)$ 在式（15）中的记号，并确定一个常数 $1 < C \in 2^{\mathcal{O}(k \log k)}$，该常数仅取决于 k，使得 $2^{\mathcal{O}(k \log k)}$ 项在式（15）中的上界为 C。这样，我们可以将不等式重新写为：

$$\Phi(\mathcal{T}) - \Phi(\mathcal{T}') \geqslant \sum_{t \in W} \mathsf{height}_T(t) - C \cdot \left(|W| + \sum_{t \in \mathtt{App}(W)} \mathsf{height}_T(t) \right) \cdot \log n. \tag{16}$$

现在我们来考察 公式（16）右侧中由一次递归调用引入的项所作出的贡献 $\mathsf{getUnbalanced}(T_r,\ j)$：加入到 P 中的有根路径 W 以及P找到的子树。其值等于

$$\sum_{t \in P} \mathsf{height}(t) - C \cdot \left(|P| + \sum_{i=1}^{b} \mathsf{height}(T_i) \right) \cdot \log n.$$

现在我们还将用于 c的值设为 引理 6.2 中用来定义序列 $h_1, \ldots\ldots, h_a$和$n_1, \ldots\ldots, n_a$ 的 $c := 20C \log n$(注意，对于充分大的 n，它小于 N，这正如所需)；作出这一选择的原因很快就会明了。明确这一点后，我们可以陈述以下命题：

命题 6.3。 以下不等式成立：

$$C \cdot \left(|P| + \sum_{i=1}^{b} \mathsf{height}(T_i) \right) \cdot \log n \leqslant \frac{1}{2} \sum_{t \in P} \mathsf{height}(t)$$

命题证明。 回想一下， $h_j = c \cdot h_{j+1}$、 $h_j \geqslant 1$ 和 $\frac{n_j}{n_{j+1}} = h_{j+1}$因此，我们可以推出以下界：

$$\begin{aligned} C \cdot \left(|P| + \sum_{i=1}^{b} \mathsf{height}(T_i) \right) \cdot \log n &\overset{(13)}{\leqslant} C \left(|P| + 2|P| \left(h_{j+1} + \frac{n_j}{n_{j+1}} \right) \right) \cdot \log n \\ &\leqslant C(|P| \cdot h_{j+1} + 2|P|(h_{j+1} + h_{j+1})) \cdot \log n \\ &= 5C \cdot |P| \cdot h_{j+1} \cdot \log n. \end{aligned}$$

另一方面，我们有

$$\sum_{t \in P} \mathsf{height}(t) \overset{(14)}{\geqslant} \frac{|P| \cdot h_j}{2} = \frac{|P| \cdot c \cdot h_{j+1}}{2} = 10C \cdot |P| \cdot h_{j+1} \cdot \log n$$

由此可得结论。 ◁

对 断言 6.3 中所有路径 P对应于 $\mathsf{getUnbalanced}$的调用，将不等式求和后，我们立即得到以下断言：

断言 6.4. 以下不等式成立：

$$C \cdot \left(|W| + \sum_{t \in \mathtt{App}(W)} \mathsf{height}_T(t) \right) \cdot \log n \leqslant \frac{1}{2} \sum_{t \in W} \mathsf{height}(t).$$

将其应用于 公式（16），可得 $\Phi(\mathcal{T}) - \Phi(\mathcal{T}') \geqslant \frac{1}{2}\sum_{t\in W}$ 高度(t)。根据性质 refine(W)，其运行时间的界为：(RT1)

$$2^{\mathcal{O}(k^9)} \cdot \left(|W| + \sum_{t\in \texttt{App}(W)} \mathsf{height}(t) \right) \cdot \log n + \mathcal{O}(\max(\Phi(\mathcal{T}) - \Phi(\mathcal{T}'), 1)).$$

（和之前一样，我们用到了这样的事实：$\log |V$（T）$| \leqslant \mathcal{O}$（log $n + k \log k$）。）然而，根据断言 6.4 我们有：

$$C \cdot \left(|W| + \sum_{t\in \texttt{App}(W)} \mathsf{height}_T(t) \right) \cdot \log n \leqslant \frac{1}{2} \sum_{t\in W} \mathsf{height}(t) \leqslant \Phi(\mathcal{T}) - \Phi(\mathcal{T}').$$

由此可得，运行时间的界为 $2^{\mathcal{O}(k^9)}(\Phi(\mathcal{T}) - \Phi(\mathcal{T}'))$。通过使用引理 3.5来实现 getUnbalanced$(T,\ 1)$，其总运行时间可轻易界定为 $\mathcal{O}(|W|)$；而该时间同样可界定为 $\mathcal{O}(\Phi(\mathcal{T}) - \Phi(\mathcal{T}'))$。因此，如果没有发生缩减过程，高度改进操作中循环的一次运行时间也可界定为 $2^{\mathcal{O}(k^9)}(\Phi(\mathcal{T}) - \Phi(\mathcal{T}'))$。

现在我们来界定缩减过程的运行时间。如果发生缩减，则有 $|V(T)| \geqslant B \cdot n^3$。我们知道 $\Phi(\mathcal{T}) \geqslant |V(T)|$。运行定理 5.3 中的算法，在 T 上使用参数 k，并对返回的树分解进行后处理，使其成为二叉树分解，需要 $2^{\mathcal{O}(k^3)} \cdot |V(T)|$ 时间，并产生一个 $\mathcal{T}^\star = (T^\star,$ 袋$^\star)$，满足 $\Phi(\mathcal{T}^\star) \leqslant \mathcal{O}(Bn^2)$。然后，用 $\mathcal{T}^\star$ 重置细化数据结构，所需时间为 $2^{\mathcal{O}(k^8)} \cdot |V(T)| = 2^{\mathcal{O}(k^8)} \cdot \mathcal{O}(Bn^2)$。

由于 $\Phi(\mathcal{T}) \geqslant Bn^3$ 和 $\Phi(\mathcal{T}^\star) \leqslant \mathcal{O}(Bn^2)$，对于足够大的$n \in \Omega(1)$，我们有 $\mathcal{O}(\Phi(\mathcal{T})) = \mathcal{O}(\Phi(\mathcal{T}) - \Phi(\mathcal{T}'))$。因此，缩减过程的运行时间可界定为 $2^{\mathcal{O}(k^8)} \cdot |V(T)| \leqslant 2^{\mathcal{O}(k^8)}\Phi(\mathcal{T}) = 2^{\mathcal{O}(k^8)}(\Phi(\mathcal{T}) - \Phi(\mathcal{T}'))$。

由于 $\Phi(T)$ 在主循环的每次迭代后都严格递减，因此它必定会在某个时刻终止。由于缩减过程和高度改进的运行时间都可以界定为 $2^{\mathcal{O}(k^9)}$ 乘以势函数的下降量，因此主循环的完整单次迭代也满足同样的界。由此，improveHeight运行时间的最坏情况界improveHeight 已由引理 6.1 中的陈述得证。

回顾一下，$h_1 = 2^{\mathcal{O}(\sqrt{\log N \text{对数} c})}$。令 $c = 2^{\mathcal{O}(k \text{对数} k)}$ 对数 n 和 $N \leqslant B \cdot n^3$，其中 $B = 2^{\mathcal{O}(k \text{对数} k)}$，因此对数 $c = \mathcal{O}(k$ 对数 $k +$ 对数 对数 $n)$，且对数 $N = \mathcal{O}(k$ 对数 $k +$ 对数 $n)$。因此，

$$h_1 = 2^{\mathcal{O}(\sqrt{\log N \log c})} = 2^{\mathcal{O}(k \log k \sqrt{\log n \log\log n})}.$$

由于 improveHeight 仅当树分解的高度降至 h_1以下时才会终止，因此引理 6.1陈述中的高度条件也成立。此外，由于 improveHeight 仅通过 refine 更新树分解，或将其替换为宽度至多为 $k < \ell$的新分解，因此可以保证，每当improveHeight 在宽度至多为 $\mathcal{T}$的树分解 ℓ上调用时，所得分解 $\mathcal{T}'$ 的宽度也至多为 ℓ。最后，显然 $V(T') \leqslant k^{\mathcal{O}(k)} \cdot n^3$ 和 $\Phi(\mathcal{T}') \leqslant \Phi(\mathcal{T})$。引理引理 6.1的证明。

7 引理证明3.9

完成所有必要的准备后，我们已准备好证明引理 3.9。为方便起见，我们重述其内容。

引理 3.9。 存在一种数据结构，对于一个在初始化时固定的整数 $k \in \mathbb{N}$，以及一个通过边插入和删除进行更新的动态图 G，能够维护一个带注释的树分解$(T$、袋、边$)$，其宽度至多为 $6k+5$，并使用前缀重建更新；前提是$\mathrm{tw}(G) \leqslant k$ 始终成立。更准确地说，在任意时刻，保证该图的树宽至多为 k，且该数据结构包含一个宽度至多为 $6k+5$的 G 带注释的树分解。该数据结构可在 k 上初始化，初始图 G 是一个具有 n 个顶点且无边的图，初始化时间为 $\cdot n)$，此后每次更新：

- 返回用于修改树分解的前缀重建更新序列；并且
- 的摊销时间为 $2^{\mathcal{O}(k^9+k\text{对数 }k\cdot\sqrt{\text{对数 }n\overline{\text{对数对数 }n)}}}$。

同样，与 第 5 节中一样，我们固定 $\ell = 6k+5$ 并取 $\Phi :=\Phi_\ell$ 和 $g :=g_\ell$，其定义见 第 5.1 节。首先，我们将描述如何实现该数据结构；然后，我们将界定一系列边更新的运行时间。

7.1 数据结构

为了初始化一个动态变化的带注释的树分解 $\mathcal{T} =(T$，袋，边$)$，我们实例化以下 $(6k+6)$-前缀重建数据结构：

- $\mathbb{D}$：开销为 $2^{\mathcal{O}(k^8)}$ 的 引理 5.1中的细化数据结构，此外还支持 improveHeight (引理 6.1)；
- $\mathbb{H}$：开销为 $\mathcal{O}(1)$ 的数据结构，依据 引理 3.5，使我们能够针对每个顶点 $v \in V(G)$查询包含 $\mathsf{top}(v) \in V(T)$ 的顶层袋 v；并针对每个 $t \in V(T)$，查询 t 在 $V(T)$中的高度。

两个数据结构存储相同的分解（T，袋，边）：$\mathbb{D}$ 执行的所有前缀重建更新都会转发给 $\mathbb{H}$。初始化后以及每次查询后，我们都维持以下不变量：$(T$，袋，边$)$包含 G 的一个带注释的树分解，其宽度至多为 $\ell = 6k+5$，高度至~~多为~~ $2^{\mathcal{O}(k\log k\sqrt{\log n\log\log n})}$，规模至多为 $k^{\mathcal{O}(k)}\cdot n^3$。每次查询后，我们都会返回 $\mathbb{D}$执行的前缀重建更新序列。

对于空图的结构初始化，我们可以将 $\mathcal{T} =(T$，袋，边$)$初始化为一棵 T完全二叉树（高度为 $\mathcal{O}(\log n)$），含有 n 个节点，并且每个袋都包含 $V(G)$的一个不同顶点。显然，对于每个 $t \in V(T)$，都有 $\mathsf{edges}(t) =\emptyset$。用它来初始化 $\mathbb{D}$和 $\mathbb{H}$需要 $2^{\mathcal{O}(k^8)}\cdot n$时间。值得注意的是，这样的分解 $\mathcal{T}$满足 $\Phi(\mathcal{T}) \leqslant \mathcal{O}(kn)$，因为 T中一个节点的平均高度为 $\mathcal{O}(1)$，且 $g(1) \in \mathcal{O}(k)$。

首先，假设我们要添加一条边 uv 到 G。令 G' 表示在图 G 中添加边 uv 后得到的图。为此，我们必须先确保新边满足边条件，即 u 和 v 都属于 $\mathcal{T}$ 的同一个袋。令 $t_u =\mathsf{top}(u)$ 和 $t_v =\mathsf{top}(v)$ 分别表示 u 和 v 在 T 中的顶层袋。如果 $v \in \mathsf{bag}(t_u)$ 或 $u \in \mathsf{bag}(t_v)$，那么边条件已经满足；不过，我们仍需将新添加的边 uv 更新到函数 edges 中。回顾 edges 的定义，边 uv 应恰好包含在一个集合中：包含 u 和 v 的最高层袋 t 对应的集合 $\mathsf{edges}(t)$。容易验证，这个袋实际上是 t_u 和 t_v 中高度较小的那个（高度以 T 为准）。不失一般性，假设它是 t_v。我们通过对从 T 的根到 t_v 的路径执行一次简单的前缀重建更新，将 uv 加入 $\mathsf{edges}(t_v)$。重建的前缀规模至多为 $\mathsf{height}(T)$；因此，该更新耗时 $2^{\mathcal{O}(k^8)}\cdot\mathsf{height}(T)$，并生成 G' 的树分解 $\mathcal{T}'$。此处树分解的结构保持不变，因此更新保留了不变量。

现在假设 $v \notin \mathsf{bag}(t_u)$ 且 $u \notin \mathsf{bag}(t_v)$；在这种情况下，我们必须扩展分解中的一些袋，以满足边条件。令 P_u 和 P_v 分别表示从 T 的根到 t_u 和 t_v 的路径。随后，分解的更新分两步进行：首先，将 v 加入所有满足 $t \in P_u \cup P_v$ 的袋 $\mathsf{bag}(t)$ 中，然后才将边 uv 加入适当的集合 edges。前一步通过获取一个弱描述 $\widehat{u}$ 来完成，该描述对应于一次将 v 加入所需袋中的前缀重建更新；接着调用 $\bar{u} := \mathsf{strengthen}(\widehat{u})$（引理 3.7），再应用由此得到的前缀重建操作 $\bar{u}$。随后，按照前一种情况的方法，将边 uv 插入适当的集合 $\mathsf{edges}(t)$。注意，以上所有操作都可以在 $2^{\mathcal{O}(k^8)} \cdot \mathsf{height}(T)$ 时间内完成。

现在，$\mathcal{T}' = (T, \text{袋}', \text{边}')$ 是 G' 的正确带注释的树分解，但将 u 加入 $P_u \cup P_v$ 中的袋可能会使分解的宽度增至 $\ell + 1 = 6k + 6$，从而破坏宽度不变量。为了解决这一问题，我们调用 $\mathsf{refine}(P_u \cup P_v)$。注意，$P_u \cup P_v$ 覆盖了所有规模为 $\ell + 2 = 6k + 7$ 的袋，因此该调用满足引理 5.1 的前提条件，而 refine 生成的 G' 的带注释树分解 $\mathcal{T}''$ 的宽度至多为 ℓ（条件(WIDTH)）。然而，$\mathcal{T}''$ 此时的高度或规模可能过大；我们通过调用高度改进（improveHeight）来解决这一问题，得到 G' 的树分解 $\mathcal{T}'''$，其宽度至多为 ℓ，高度至多为 $2^{\mathcal{O}(k \log k \sqrt{\log n \log\log n})}$，规模至多为 $k^{\mathcal{O}(k)} \cdot n^3$。最终的树分解 $\mathcal{T}'''$ 满足所有规定的不变量。

删除 uv 中的一条边要简单得多：我们无需改变树结构 T，也无需修改树分解中各袋的内容。不过，我们仍需通过一次前缀重建更新，从适当的集合 $\mathsf{edges}(t)$ 中移除 uv，其中 $t \in V(T)$。与之前一样，定位适当的$t \in V(T)$ 的方法如下：设 $t_u =$ 为 $\mathsf{top}(u)$，$t_v =$ 为 $\mathsf{top}(v)$；根据 t_u 还是 t_v 的高度较小，uv 属于 $\mathsf{edges}(t_u)$ 或 $\mathsf{edges}(t_v)$。不失一般性地，假设 $uv \in$ 属于 $\mathsf{edges}(t_v)$。我们通过重建从 T 的根到 t_v的路径，构造一次前缀重建更新，从 $\mathsf{edges}(t_v)$ 中移除 uv 。

7.2 复杂度分析

现在，我们将为该数据结构的摊还运行时间给出一个界。如前所述，我们从一个无边图以及它的一个分解 $\mathcal{T} = (T, \text{袋}, \text{边})$开始，其势函数为$\Phi(\mathcal{T}) = \mathcal{O}(kn)$。

我们只分析边插入的情况；边删除的分析更为直接。计算 P_u，P_v 并执行初始的前缀重建更新所需的时间为

$$2^{\mathcal{O}(k^8)} \cdot \mathsf{height}(T) \leqslant 2^{\mathcal{O}(k^8)} \cdot 2^{\mathcal{O}(k \log k \sqrt{\log n \log\log n})}. \tag{17}$$

设 G' 为添加边 uv 后的图 G，而 $\mathcal{T}' = (T, \text{袋}', \text{边}')$ 为初始前缀重建更新后 G' 的树分解。（需要强调的是，分解 $\mathcal{T}$ 和 $\mathcal{T}'$ 具有相同的形状 T。）如果没有将 u 添加到任何袋中，那么 $\mathcal{T}'$ 的势函数没有增加，整个查询耗时 $2^{\mathcal{O}(k \text{对数 } k \sqrt{\text{对数 } n} \text{对数对数 } n)}$。现在假设 u 已被添加到一些新袋中。回想一下，u 至多被添加到 $\mathcal{O}(\text{高度}(T))$ 个新袋中，使得每个新袋的规模增至至多 $\ell + 2 = 6k + 7$；其他袋的规模保持不变。因此，

$$\begin{aligned} \Phi(\mathcal{T}') - \Phi(\mathcal{T}) &\leqslant g(6k+7) \cdot \mathcal{O}(\mathsf{height}(T)^2) \leqslant 2^{\mathcal{O}(k \log k)} \cdot \mathsf{height}(T)^2 \\ &\leqslant 2^{\mathcal{O}(k \log k \sqrt{\log n \log\log n})}. \end{aligned} \tag{18}$$

令 $\mathcal{T}''$ 表示通过对 $\mathsf{refine}(T_{\mathrm{pref}})$ 应用于 $\mathcal{T}'$ 并使用前缀 T_{pref} 和$|T_{\mathrm{pref}}| \leqslant \mathcal{O}(\text{高度}(T))$得到的分解。回顾界定势函数变化的性质 (POT)：

$$\Phi(\mathcal{T}') - \Phi(\mathcal{T}'') \geqslant \sum_{t \in T_{\mathrm{pref}}} \mathsf{height}_T(t) - 2^{\mathcal{O}(k \log k)} \cdot \left(|T_{\mathrm{pref}}| + \sum_{t \in \mathtt{App}(T_{\mathrm{pref}})} \mathsf{height}_T(t) \right) \cdot \log |V(T)|.$$

因此，

$$\begin{aligned}
\Phi(\mathcal{T}'') - \Phi(\mathcal{T}') &\leqslant - \sum_{t \in T_{\mathrm{pref}}} \mathsf{height}_T(t) + 2^{\mathcal{O}(k \log k)} \cdot \left(|T_{\mathrm{pref}}| + \sum_{t \in \mathtt{App}(T_{\mathrm{pref}})} \mathsf{height}_T(t) \right) \cdot \log |V(T)| \\
&\leqslant 2^{\mathcal{O}(k \log k)} \cdot |T_{\mathrm{pref}}| \cdot \mathsf{height}(T) \cdot \log |V(T)| \\
&\leqslant 2^{\mathcal{O}(k \log k)} \cdot \mathsf{height}(T)^2 \cdot \log n \\
&\leqslant 2^{\mathcal{O}(k \log k \sqrt{\log n \log \log n})}.
\end{aligned} \tag{19}$$

这里，我们利用了以下事实：$|T_{\mathrm{pref}}| \leqslant \mathcal{O}$（高度（$T$））， |App（$T_{\mathrm{pref}}$）| $\leqslant |T_{\mathrm{pref}}| + 1$ 且 log |V（T）| $\leqslant \mathcal{O}$（$k \log k + \log n$）（由不变量以及 T是二叉的事实可得）。

最后，设 $\mathcal{T}'''$ 为对 $\mathcal{T}''$运行高度改进操作（improveHeight）后得到的最终树分解。由于 improveHeight 从不增加势值，我们有Φ（$\mathcal{T}'''$） $\leqslant \Phi$（$\mathcal{T}''$）。由此可得

$$\Phi(\mathcal{T}''') - \Phi(\mathcal{T}) \leqslant 2^{\mathcal{O}(k \log k)} \cdot \mathsf{height}(T)^2 \cdot \log n \leqslant 2^{\mathcal{O}(k \log k \sqrt{\log n \log \log n})}.$$

关于边插入的运行时间，回顾性质 (RT1) 可知，在 $\mathcal{T}'$ 上执行细化操作的最坏情况时间为

$$\begin{aligned}
& 2^{\mathcal{O}(k^9)} \left(|T_{\mathrm{pref}}| + \sum_{t \in \mathtt{App}(T_{\mathrm{pref}})} \mathsf{height}(t) + \max(\Phi(\mathcal{T}') - \Phi(\mathcal{T}''), 0) \right) \\
\leqslant \;& 2^{\mathcal{O}(k^9)} \left(\mathsf{height}(T)^2 + \max(\Phi(\mathcal{T}') - \Phi(\mathcal{T}''), 0) \right) \\
\overset{(19)}{\leqslant} \;& 2^{\mathcal{O}(k^9)} \left(\mathsf{height}(T)^2 + (\Phi(\mathcal{T}') - \Phi(\mathcal{T}'')) + 2^{\mathcal{O}(k \log k)} \cdot \mathsf{height}(T)^2 \cdot \log n \right) \\
\leqslant \;& 2^{\mathcal{O}(k^9)} \left(2^{\mathcal{O}(k \log k \sqrt{\log n \log \log n})} + \Phi(\mathcal{T}') - \Phi(\mathcal{T}'') \right);
\end{aligned} \tag{20}$$

立即　随后进行高度改进，其根据引理 6.1 的最坏情况时间为

$$2^{\mathcal{O}(k^9)} \cdot (\Phi(\mathcal{T}'') - \Phi(\mathcal{T}''')) + \mathcal{O}(1). \tag{21}$$

因此，总运行时间受以下各式之和的界定：公式 (17)、(20) 和(21)：

$$\begin{aligned}
& 2^{\mathcal{O}(k^9 + k \log k \sqrt{\log n \log \log n})} + 2^{\mathcal{O}(k^9)} (\Phi(\mathcal{T}') - \Phi(\mathcal{T}''')) \\
\overset{(18)}{\leqslant} \;& 2^{\mathcal{O}(k^9 + k \log k \sqrt{\log n \log \log n})} + 2^{\mathcal{O}(k^9)} (\Phi(\mathcal{T}) - \Phi(\mathcal{T}''')).
\end{aligned}$$

由于每次更新使势值最多增加 $2^{\mathcal{O}(k \log k \sqrt{\log n \log \log n})}$，因此，正如所述，每次更新的摊销时间复杂度为 $2^{\mathcal{O}(k^9 + k \log k \sqrt{\log n \log \log n})}$。

8 结论

我们提出了一种用于动态树宽问题的数据结构，其摊销更新时间为 $2^{\mathcal{O}_k(\sqrt{\log n}\log\log n)}$。一个显而易见的开放问题是如何改进这一复杂度。对当前方法进行某些优化，或许可以将指数中的 $\sqrt{\log\log n}$ 因子去掉，但 $2^{\mathcal{O}_k(\sqrt{\log n})}$ 形式的复杂度似乎是第 第 6 节节所述递归方法所固有的。不过，我们猜想，对于每个固定的 k，都应当能够实现关于 n 呈多对数的更新时间，即 $\log^{\mathcal{O}_k(1)} n$ 形式，甚至可能达到 $\mathcal{O}_k(\log^c n)$ 形式，其中 c 是某个通用常数。最终目标是更接近 Bodlaender 在 $k=2$ [12] 情形下达到的 $\mathcal{O}(\log n)$ 界。

除此之外，我们希望我们的结果能够为参数化动态数据结构的设计开辟新的方向。更确切地说，树分解上的动态规划被用作算法设计中多种不同技术的构建模块，因此 定理 1.1 可能有助于设计这些技术的动态对应形式。以下是一些可能的方向。

- 定理 1.1 给出了 Courcelle 定理最基本表述的动态变体。由 Arnborg 等人提出的优化表述和计数表述的动态变体，似乎只需将定理 1.1的数据结构与适当构造的自动机结合起来即可得到。值得研究的是，定理 1.1是否也能用于更一般的情形，即在有界树宽图上处理关于 CMSO_2 查询的问题，例如查询枚举问题；参见，例如 [9, 38]。

- 双维性 和 移位 是用于设计平面图（或更一般地，排除某个固定图子式的图）上的参数化算法的两种基本技术；参见 [24, 第 7 节。7]。由于这两种技术都基于在有界树宽的图上高效求解问题，因此可以研究能否利用 定理 1.1 开发相应的动态版本。

- 与上文相关，一个值得探讨的问题是，能否利用定理 1.1 设计 Baker 技术的动态版本 [10]。这样便能构建动态数据结构，用于维护诸如 独立集或支配集 等问题在平面图上的近似方案。

- 无关顶点技术 是参数化算法中的经典原理，其做法是反复从图中删除顶点，同时保持问题的答案不变，直到图具有有界树宽，便可通过动态规划直接求解。尽管颇具挑战性，但利用无关顶点技术得到的某些算法似乎并非不可能衍生出动态版本。一个具体的候选问题是平面图上的不相交路径问题；该问题具有一条相对简单且广为人知的无关顶点规则 [1]

- 元核化是由 Bodlaender 等人提出的 [14], 一种强大的元技术，用于为拓扑受限图上的参数化问题获得小核。元核化的基本概念是突出部：图中的一部分，它所诱导的子图具有有界树宽，并通过一个规模有界的接口与图的其余部分相连。元核化所得到的核化算法使用基于突出部替换的归约规则：找到一个大突出部，利用基于树分解的动态规划对其进行分析，再用一个功能相同的小构件替换它。若能在某些基本情形中，将定理 1.1

与元核化结合使用，以获得用于在动态图中维护小核的数据结构，那将很有意思。平面图上的支配集或许是首先值得研究的问题，因为它存在一种简单的核化算法，且该算法早于（并启发了）元核化 [2]。值得注意的是，动态核化已作为一种获取动态参数化数据结构的方法得到研究 [3, 37]。

致谢。 作者感谢 Anna Zych-Pawlewicz 在本项目开发过程中提出的许多有见地的讨论意见。

参考文献

[1] Isolde Adler、Stavros G. Kolliopoulos、Philipp Klaus Krause、Daniel Lokshtanov、Saket Saurabh 和 Dimitrios M. Thilikos。《平面图不相交路径问题中的无关顶点》。《组合理论杂志》*B*辑，122：815–843，2017年。[2] Jochen Alber、Michael R. Fellows 和 Rolf Niedermeier。《支配集的多项式时间数据归约》。*J.ACM*，51(3)：363–384，2004年。[3] Josh Alman、Matthias Mnich 和 Virginia Vassilevska Williams。《动态参数化问题与算法》。《*ACM*算法汇刊》，16(4)：45:1–45:46，2020年。[4] Stephen Alstrup、Jacob Holm、Kristian de Lichtenberg 和 Mikkel Thorup。《利用顶树维护全动态树中的信息》。《*ACM*算法汇刊》，1(2)：243–264，2005年。[5] Antoine Amarilli、Pierre Bourhis、Stefan Mengel 和 Matthias Niewerth。《在树上实现可处理的组合复杂度与高效更新的枚举》。*CoRR*，abs/1812.09519，2018年。[6] Antoine Amarilli、Pierre Bourhis、Stefan Mengel 和 Matthias Niewerth。《在树上实现可处理的组合复杂度与高效更新的枚举》。载于第*38*届 *ACM SIGMOD-SIGACT-SIGAI* 数据库系统原理研讨会，*PODS 2019*，第89–103页。ACM，2019年。[7] Stefan Arnborg、Derek C. Corneil 和 Andrzej Proskurowski。《在 k-树中的嵌入查找复杂度》。*SIAMJ.Alg.Disc. Meth.*，8：277–284，1987年。[8] Stefan Arnborg、Jens Lagergren 和 Detlef Seese。《树可分解图上的简单问题》。《算法杂志》，12(2)：308–340，1991年。[9] Guillaume Bagan。《树可分解结构上的 MSO 查询可在线性延迟下计算》。载于第*20*届计算机科学逻辑国际研讨会，*CSL 2006*，第4207卷《计算机科学讲义》，第167–181页。施普林格，2006年。[10] Brenda S. Baker。《平面图上 NP 完全问题的近似算法》。*J.ACM*，41(1)：153–180，1994年。[11] Patrick Bellenbaum 和 Reinhard Diestel。《关于树分解的两个简短证明》。*Comb.Probab.Comput.*，11(6)：541–547，2002年。[12] Hans L. Bodlaender。《树宽为2的图的动态算法》。载于第*19*届图论概念国际研讨会，*WG 1993*，第790卷《计算机科学讲义》，第112–124页。施普林格，1993年。

[13]Hans L. Bodlaender。《寻找小树宽树分解的线性时间算法》。*SIAMJ.Comput.*，25(6)：1305–1317，1996年。[14] Hans L. Bodlaender、Fedor V. Fomin、Daniel Lokshtanov、Eelko Penninkx、Saket Saurabh 和 Dimitrios M. Thilikos。《（元）核化》。*J. ACM*，63(5)：44:1–44:69，2016年。[15] Hans L. Bodlaender 和 Torben Hagerup。《有界树宽的最优加速并行算法》。*SIAMJ.Comput.*，27(6)：1725–1746，1998年。[16] Hans L. Bodlaender 和 Ton Kloks。《图的路径宽与树宽的高效构造性算法》。《算法杂志》，21(2)：358–402，1996年。[17] Mikołaj Bojańczyk 和 Michał Pilipczuk。《在 MSO 中优化树分解》。*Log. MethodsComput. Sci.*，18(1)，2022年。[18] Gerth Stølting Brodal 和 Rolf Fagerberg。《稀疏图的动态表示》。载于算法与数据结构，第六届国际研讨会，*WADS*，第1663卷《计算机科学讲义》，第342–351页。施普林格，1999年。[19] Jiehua Chen、Wojciech Czerwiński、Yann Disser、Andreas Emil Feldmann、Danny Hermelin、Wojciech Nadara、Marcin Pilipczuk、Michał Pilipczuk、Manuel Sorge、Bartlomiej Wróblewski 和 Anna Zych-Pawlewicz。《用于检测长路径和环的高效全动态消去森林》。载于*2021*年 *ACM-SIAM* 离散算法研讨会，*SODA 2021*，第796–809页。SIAM，2021年。[20] Jiehua Chen、Wojciech Czerwiński、Yann Disser、Andreas Emil Feldmann、Danny Hermelin、Wojciech Nadara、MichałPilipczuk、Marcin Pilipczuk、Manuel Sorge、Bartlomiej Wróblewski 和 Anna Zych-Pawlewicz。《用于检测长路径和环的高效全动态消去森林》。*CoRR*，abs/2006.00571，2020年。[21] Robert F. Cohen、Sairam Sairam、Roberto Tamassia 和 Jeffrey Scott Vitter。《有界树宽图上优化问题的动态算法》。载于第三届整数规划与组合优化会议，*IPCO 1993*，第99–112页。CIACO，1993年。[22] Bruno Courcelle。《图的二阶单子逻辑：I. 有限图的可识别集合》。*Inf.Comput.*，85(1)：12–75，1990年。[23] Bruno Courcelle 和 Joost Engelfriet。《图结构与单子二阶逻辑——语言理论方法》，第138卷，《数学及其应用百科全书》。剑桥大学出版社，2012年。[24] Marek Cygan、Fedor V. Fomin、Łukasz Kowalik、Daniel Lokshtanov、Dániel Marx、Marcin Pilipczuk、Michał Pilipczuk 和 Saket Saurabh。《参数化算法》。施普林格，2015年。[25] Rodney G. Downey 和 Michael R. Fellows。《参数化复杂性基础》。《计算机科学文本》。施普林格，2013年。[26] Zdenek Dvořák、Martin Kupec 和 Vojtech Tůma。《有界树深度图上用于 MSO 性质的动态数据结构》。载于第*22*届欧洲算法年会，*ESA 2014*，第8737卷《计算机科学讲义》，第334–345页。施普林格，2014年。

[27] Zdenek Dvořák 和 Vojtech Tůma。用于在稀疏图中计数子图的动态数据结构。载于第*13*届国际算法与数据结构研讨会，*WADS 2013*，收入《计算机科学讲义》第8037卷，第304–315页。施普林格，2013年。[28] David Eppstein、Zvi Galil、Giuseppe F. Italiano 和 Thomas H. Spencer。Bodlaender 与 Courcelle 定理的对数空间版本。载于第*51*届 *IEEE* 计算机科学基础年会，*FOCS 2010*，第143–152页。IEEE 计算机学会，2010年。[29] David Eppstein、Zvi Galil、Giuseppe F. Italiano 和 Thomas H. Spencer。基于分隔集的稀疏化。I. 平面性测试与最小生成树。《计算机与系统科学杂志》，52(1):3–27，1996年。[30] Uriel Feige、MohammadTaghi Hajiaghayi 和 James R. Lee。最小权顶点分隔器的改进近似算法。*SIAM J. Comput.*，38(2):629–657，2008年。[31] Jörg Flum 和 Martin Grohe。参数化复杂性理论。《理论计算机科学丛书》。EATCS 丛书。施普林格，2006年。[32] Greg N. Frederickson。在 k端点图中动态维护正则性质。*Algorithmica*，22(3):330–350，1998年。[33] Gramoz Goranci、Harald Räcke、Thatchaphol Saranurak 和 Zihan Tan。扩张图层级及其在动态图算法中的应用。载于*2021*年 *ACM-SIAM* 离散算法研讨会，*SODA 2021*，第2212–2228页。SIAM，2021年。[34] Alejandro Grez、Filip Mazowiecki、Michał Pilipczuk、Gabriele Puppis 和 Cristian Riveros。定时自动机接受性的动态数据结构。*Algorithmica*，84(11):3223–3245，2022年。[35] Martin Grohe 和 Stephan Kreutzer。算法元定理的方法。载于 Martin Grohe 和 Johann A. Makowsky 编，有限组合学中的模型论方法——*AMS-ASL*联合专题会议，收入当代数学第558卷，第181–206页。美国数学会，2009年。[36] Neil Immerman。描述复杂性。计算机科学研究生教材。施普林格，1999年。[37] Yoichi Iwata 和 Keigo Oka。参数化问题的快速动态图算法。载于第*14*届斯堪的纳维亚算法理论研讨会及专题讨论会，*SWAT 2014*，收入《计算机科学讲义》第8503卷，第241–252页。施普林格，2014年。[38] Wojciech Kazana 和 Luc Segoufin。树上的单子二阶查询枚举。《*ACM*计算逻辑汇刊》，14(4):25:1–25:12，2013年。[39] Jon M. Kleinberg 和 Éva Tardos。算法设计。Addison-Wesley，2006年。[40] Tuukka Korhonen。一种用于树宽的单指数时间2-近似算法。载于第*62*届 *IEEE* 计算机科学基础年会，*FOCS 2021*，第184–192页。IEEE，2021年。[41]Tuukka Korhonen 和 Daniel Lokshtanov。改进的树宽参数化算法。*CoRR*，abs/2211.07154，2022年。已被 STOC 2023 接收。[42] Jens Lagergren。阻碍与交织结构规模的上界。《组合理论杂志》*B*辑，73(1):7–40，1998年。

[43] Konrad Majewski、Michał Pilipczuk 和 Marek Sokołowski。在反馈顶点数有界的动态结构上维护 CMSO_2 性质。载于第*40*届国际计算机科学理论方面研讨会，*STACS 2023*，收入*LIPIcs*第254卷，第46:1–46:13页。Schloss Dagstuhl — Leibniz-Zentrum für Informatik，2023年。[44] Johann A. Makowsky。Feferman-Vaught定理的算法应用。《纯粹与应用逻辑年刊》，126(1-3):159–213，2004年。[45] Matthias Niewerth。树上的 MSO 查询：使用森林代数在更新期间枚举答案。载于第*33*届 *ACM/IEEE*计算机科学逻辑年会，*LICS 2018*，第769–778页。ACM，2018年。[46] Jędrzej Olkowski、Michał Pilipczuk、Mateusz Rychlicki、Karol Węgrzycki 和 Anna Zych-Pawlewicz。参数化字符串问题的动态数据结构。载于第*40*届国际计算机科学理论方面研讨会，*STACS 2023*，收入*LIPIcs*第254卷，第50:1–50:22页。Schloss Dagstuhl — Leibniz-Zentrum für Informatik，2023年。[47] Sang-il Oum。快速近似秩宽度和团宽度。《*ACM*算法汇刊》，5(1):10:1–10:20，2008年。[48] Sang-il Oum 和 Paul D. Seymour。近似团宽度和分支宽度。《组合理论杂志》*B*辑，96(4):514–528，2006年。[49] Michał Pilipczuk。计算树分解。载于 Fedor V. Fomin、Stefan Kratsch 和 Erik Jan van Leeuwen 编，树宽、核与算法——献给 *Hans L. Bodlaender 60*岁生日的论文集，收入《计算机科学讲义》第12160卷，第189–213页。施普林格，2020年。[50] Neil Robertson 和 Paul D. Seymour。图小子 II. 树宽的算法方面。《算法杂志》，7(3):309–322，1986年。[51] Neil Robertson 和 Paul D. Seymour。图小子 V. 排除平面图。《组合理论杂志》*B*辑，41(1):92–114，1986年。[52] Daniel D. Sleator 和 Robert E. Tarjan。用于动态树的数据结构。《计算机与系统科学杂志》，26(3):362–391，1983年。[53] Robin Thomas。树宽的 Menger型性质：有限情形。《组合理论杂志》*B*辑，48(1):67–76，1990年。[54] Yu Wu、Per Austrin、Toniann Pitassi 和 David Liu。树宽及相关问题的不可近似性。《人工智能研究杂志》，49:569–600，2014年。

动态自动机

本节将介绍一个框架，用于在树分解上通过前缀重建更新动态维护动态规划表。该框架的具体成果包括以下证明：引理 3.5、引理 3.8和引理 4.23；不过，该框架具有足够的通用性，也能用于维护任何合理的动态规划方案。

我们指出，关于动态森林上的自动机运行的维护，文献中已经有相关研究，甚至还涉及更为一般的动态枚举问题；参见……

例如 Niewerth 的工作 [45] 以及 Amarilli 等人的工作 [6] 和其中的文献讨论。特别是，本节所包含的许多（尽管并非全部）结果原则上都可以由 [5, 引理 7。3], 推导出来，但无法以黑箱方式完成，也无法得到具体的更新时间界。因此，为了完整起见，本节提供一个自包含的论述。

A.1 树分解自动机

我们的框架基于一种处理树分解的自动机概念。尽管这里针对我们的特定目的对这一概念进行了调整，但使用各种类型的自动机或动态规划过程来处理树分解这一想法，早在 Courcelle [22]的研究中就已出现，并且已得到充分研究；请参阅教科书中的相关章节 [24,25,31] 及其中的参考文献注释。因此，本节全部内容都可以视为对众所周知的结果的形式化。

在本节中，我们假设所考虑的图的所有顶点均取自一个固定的、全序的可数顶点集Ω。此外，我们假设在 RAM 模型中，可以在常数时间内对 Ω 的元素进行操作。读者可以假设 $\Omega = \mathbb{N}$。

带边界图。 我们将处理具有指定边界的图，具体定义如下。

定义 9。 带边界图是一个无向图 G，以及一个顶点子集 $\partial G \subseteq V(G)$，称为其边界，满足 G 中没有一条边的两个端点都位于 ∂G中。带边界图 G 的带边界树分解是一个三元组 $(T$、袋、边$)$，它是 G （视为普通图）的一个带注释的树分解，并且还要求 ∂G包含在根袋中。

当讨论带边界图 G 的带边界树分解（T、袋、边）时，我们将根 T 的粘合集重新定义为 ∂G，而不是空集。

假设（T、袋、边）是带边界图 G 的带边界树分解，且 x 是 T 的一个节点。那么我们称 x 诱导出带边界图 G_x 及其带边界树分解（T_x、袋$_x$、边$_x$），定义如下：若 X 是 T 中 x 的后代集合，则

$$\begin{aligned} G_x &= \left(\bigcup_{y \in X} \mathtt{bag}(y), \bigcup_{y \in X} \mathtt{edges}(y) \right); \\ \partial G_x &= \mathtt{adh}(x); \\ (T_x, \mathtt{bag}_x, \mathtt{edges}_x) &= (T, \mathtt{bag}, \mathtt{edges})|_X. \end{aligned}$$

显然，按上述方式定义的（T_x、袋$_x$、边$_x$）是 G_x 的一个带边界树分解。只有当带边界树分解（T、袋、边）可由上下文明确判定时，我们才使用上述记法。

自动机。 现在我们介绍自动机模型。

定义 10。 一个宽度为 ℓ 的（确定性）树分解自动机由组成

- 一个状态集 Q;
- 一组接受状态 $F \subseteq Q$;
- 一个初始映射 ι，将每个至多有 $\ell + 1$ 个顶点的带边界图 G 映射到一个状态$\iota(G) \in Q$；以及

- 一个转移映射 δ，将每个形式为(B, X, Y, Z, J, q', q'')的7元组映射到一个状态$\delta(B, X, Y, Z, J, q', q'') \in Q$，其中 $B \subseteq \Omega$ 是一个规模至多为 $\ell+1$, $X, Y, Z \subseteq B$, $J \in \binom{B}{2} \setminus \binom{X}{2}$, $q' \in Q$ 的集合，并且 $q'' \in Q \cup \{\bot\}$。

树分解自动机 $\mathcal{A}$ 在带边界图 G 的二叉带边界树分解(T、袋、边)上的运行，是满足以下性质的唯一标记 $\rho_{\mathcal{A}}: \quad V(T) \to Q$:

- 对于 T的每个叶节点 l，都有

$$\rho_{\mathcal{A}}(l) = \iota(G_l).$$

- 对于 T 中有一个子节点 y的每个非叶节点 x，都有

$$\rho_{\mathcal{A}}(x) = \delta(\mathtt{bag}(x), \mathtt{adh}(x), \mathtt{adh}(y), \emptyset, \mathtt{edges}(x), \rho_{\mathcal{A}}(y), \bot).$$

- 对于 T 中有两个子节点 y 和 z的每个非叶节点 x，都有

$$\rho_{\mathcal{A}}(x) = \delta(\mathtt{bag}(x), \mathtt{adh}(x), \mathtt{adh}(y), \mathtt{adh}(z), \mathtt{edges}(x), \rho_{\mathcal{A}}(y), \rho_{\mathcal{A}}(z)).$$

树分解自 mp动机 $\mathcal{A}$接受 (T、袋、边) 当且仅当 $\rho_{\mathcal{A}}(r) \in F$，其中 r是 T的根。

请注意，在上述转移中，对于只有一个子节点的节点，会将“虚拟状态” $\bot \notin Q$ 传递给转移函数，而不是传递一个状态。请注意，这使 δ 也能够识别只有一个子节点的情况。原则上，上述自动机模型也可以区分左子节点 y 和右子节点 z，并对从它们传递来的状态采取不同处理。然而，在我们的应用中绝不会出现这种情况：在所有构造出的自动机中，交换子节点 y 和 z的角色不会改变转移映射。

如果函数 ι 和 δ 能够在时间 τ内对任意参数元组求值，并且对于给定的 $q \in Q$，能够在时间 τ内判定 $q \in F$ 是否成立，我们就称树分解自动机 $\mathcal{A}$ 具有求值时间 τ。请注意，我们不 要求状态空间 Q 是有限的。事实上，在我们的大多数应用中，状态空间都是无限的，但我们能够高效地表示和操作这些状态。

我们经常会在非带边界图 G 的非带边界标注树分解上运行树分解自动机。在这种情况下，我们只需应用上述所有定义，同时将 G 视为边界为空的带边界图。

我们还将使用非确定性树分解自动机，其定义与定义 10 中的定义完全相同，只是 ι 和 δ 分别是初始关系和转移关系，而不是映射。也就是说，ι 是由形如 (G, q) 的对组成的关系，其中 G 是一个至多含有 $\ell+1$ 个顶点的带边界图，并且 $q \in Q$。类似地，δ 是由形如 $((B, X, Y, Z, J, q', q'')\ q)$ 的对组成的关系，其中 (B, X, Y, Z, J, q', q'') 是一个与转移映射定义域中的元组类似的七元组，并且 $q \in Q$。接着，对于带边界二叉树分解 (T, bag, edges)，非确定性自动机 $\mathcal{A}$ 的一次运行是对 T 的节点进行状态标记的标记 ρ，满足：对于每个叶节点 l，$(G_l, \rho(l)) \in \iota$；对于每个只有一个子节点 y 的节点 x，((bag(x), adh(x), adh(y) $\emptyset$, edges(x) $\rho(y)$ $\bot$) $\rho(x)$) $\in \delta$；对于每个有两个子节点 y 和 z 的节点 x，((bag(x), adh(x), adh(y), adh(z), edges(x) $\rho(y)$ $\rho(z)$) $\rho(x)$) $\in \delta$。注意，非确定性树分解自动机在同一个树分解上可能有多个运行。如果在 (T, bag, edges) 上存在一次接受状态的运行，即与根节点关联的状态为接受状态，则称 $\mathcal{A}$ 接受 (T, bag, edges)。

在非确定性自动机的语境中，求值时间指的是判定给定的一对是否属于关系 ι 或 δ，或判定给定状态

是否为接受状态所需的时间。注意，如果 $\mathcal{A}$ 是一个状态空间有限、状态空间为 Q的非确定性树分解自动机，那么我们可以使用标准的幂集构造将其确定化，得到一个接受相同树分解的确定性自动机 $\mathcal{A}'$。于是， $\mathcal{A}'$ 的状态空间为 2^Q。下文中，除非另有说明，所有自动机均为确定性自动机。

A.2 自动机构造

在后续章节中，我们将使用若干自动机。现在介绍四种将会用到的自动机构造。

树分解性质自动机。 我们首先构造三个非常简单的自动机，用于 引理 3.5 中维护树分解本身的性质。

引理 A.1。 对于每个 $\ell \in \mathbb{N}$，都存在树分解自动机$\mathcal{H}_\ell$、 $\mathcal{S}_\ell$、 $\mathcal{C}_\ell$，它们的宽度均为 ℓ，并满足以下性质：对于任意图 G、其带注释的二叉树分解$(T, \text{袋}, \text{边})$，若其宽度至多为 ℓ，且任取 G 中的节点 x：

- $\rho_{\mathcal{H}_\ell}(x)$ 等于高度(T_x)，
- $\rho_{\mathcal{S}_\ell}(x)$ 等于 $|V(T_x)|$，并且
- $\rho_{\mathcal{C}_\ell}(x)$ 等于 $|\texttt{cmp}(x)|$。

$\mathcal{H}_\ell$ 和 $\mathcal{S}_\ell$ 的求值时间为 $\mathcal{O}(1)$，而 $\mathcal{C}_\ell$ 的求值时间为 $\mathcal{O}(\ell)$。

证明。 每个自动机的状态集为 $\mathbb{N}$。我们为所有 $n \in \mathbb{N}$ 定义 $\max(n, \perp) = n$和 $n + \perp = n$。对于高度自动机 $\mathcal{H}_\ell$，初始映射和转移映射定义如下（此处 _ 表示任意输入值）：

$$\begin{aligned} \iota(_) &= 1 \\ \delta(_,_,_,_,_,q',q'') &= 1 + \max(q',q''). \end{aligned}$$

对于大小 自动机 $\mathcal{S}_\ell$，初始映射和转移映射定义如下 ：

$$\begin{aligned} \iota(_) &= 1 \\ \delta(_,_,_,_,_,q',q'') &= 1 + q' + q''. \end{aligned}$$

对于 $|\texttt{cmp}(x)|$ 自动机 $\mathcal{C}_\ell$，初始映射和转移映射定义如下：

$$\begin{aligned} \iota(G) &= |V(G) \setminus \partial G| \\ \delta(B,X,_,_,_,q',q'') &= q' + q'' + |B \setminus X|. \end{aligned}$$

很容易看出，这些自动机满足所需的性质。 □

CMSO_2-类型自动机。经典的 Courcelle 定理 [22]指出，存在一种算法，给定一个CMSO_2 语句 φ 和一个 n顶点图 G 以及宽度至多为 ℓ的树分解后，可以判定 $G \models \varphi$，且运行时间为 $f(\ell, \varphi) \cdot n$，其中 f 是一个可计算函数。证明 Courcelle 定理的一种方法是构造一个动态规划过程，自底向上处理给定的树分解。这个动态规划过程可以理解为定义 10 所述的树分解自动机，从而得到以下结果。该证明是对单子二阶逻辑CMSO_2-类型概念的完全标准应用，因此我们只作简要说明。

引理 A.2。对于每个整数 ℓ 和CMSO_2 语句 φ，都存在一个宽度为 ℓ 的树分解自动机$\mathcal{A}_{\ell,\,\varphi}$，它具有以下性质：对于任意图 G 及其带注释的二叉树分解(T、袋、边)，其宽度至多为 ℓ $\mathcal{A}_{\ell,\,\varphi}$ 当且仅当 $G \models \varphi$ 时，该自动机接受(T、袋、边)。求值时间的上界为 $f(\ell\varphi)$，其中 f是某个可计算函数。

证明概要。令 p 为 φ 的秩：取 φ 的量词秩与 φ 中出现的模谓词的模数两者中的最大值。考虑任意有限的 $X \subseteq \Omega$，并令 $\mathsf{CMSO}_2(X)$ 表示所有还可以将 X 的元素用作常量的 CMSO_2 语句（形式上，这些是图的签名中加入每个 $x \in X$ 作为常量后得到的 CMSO_2语句）。众所周知（例如参见 [36, 习题 6.11]），对于上述给定的 X，可以计算出一个集合 $\mathsf{Sentences}^p(X)$，其中至多包含 $g(\ell, |X|)$ 个 $\mathsf{CMSO}_2(X)$ 语句，对某个可计算的 g 而言，使得对于每个秩不超过 p 的 $\psi \in \mathsf{CMSO}_2(X)$，都存在一个 $\psi' \in \mathsf{Sentences}^p(X)$ 中的语句，与 ψ 等价，即它们在恰好相同的包含 X 的图上成立。此外，映射 $\psi \mapsto \psi'$ 是可计算的。

对于给定的带边界图 G 及其 $X = \partial G$，我们如下定义 G 的 p-类型：

$$\mathrm{tp}^p(G) = \{\psi \in \mathrm{Sentences}^p(X) \mid G \models \psi\}.$$

现在，我们构造自动机 $\mathcal{A} = \mathcal{A}_{\ell,\,\varphi}$，使得对于图 G的每个带边界二叉树分解（T，袋，边），$\mathcal{A}$ 在（T，袋，边）上的运行如下：

$$\rho_{\mathcal{A}}(x) = \mathrm{tp}^p(G_x) \qquad \text{for all } x \in V(T).$$

构造 $\mathcal{A}$时，唯一非平凡的检查是能否定义一个合适的转移映射δ。为此，只需证明：对于 T中具有子节点 y 和 z的每个节点 x，给定类型 $\mathsf{tp}^p(G_y)$ 和 $\mathsf{tp}^p(G_z)$，以及关于 x 的袋的信息（由 $\mathsf{bag}(x)$、$\mathsf{adh}(x)$、$\mathsf{adh}(y)$、$\mathsf{adh}(z)$ 和 $\mathsf{edges}(x)$ 组成），就可以计算出类型 $\mathsf{tp}^p(G_x)$；对于只有一个子节点的节点，同样如此。这可由涉及 Ehrenfeucht-Fraïssé 博弈的标准论证得出，参见 [35, 44]。 □

Bodlaender-Kloks 自动机。在 [16], Bodlaender 和 Kloks 给出了一种算法，该算法给定一个图 G、一个宽度至多为 ℓ 的 G 二叉树分解以及一个数 $k \leqslant \ell$，可在时间 $2^{\mathcal{O}(k\ell^2)} \cdot n$ 内判定 G 的树宽是否至多为 k，其中 n 是 G 的顶点数。该算法在给定的 G 树分解上进行自底向上的动态规划，对每个节点 x 计算一个由 $2^{\mathcal{O}(k\ell^2)}$ 个布尔值组成的表。直观地说，每个表项都表示：是否能够构造出一个部分树分解，使其对应于由 x 处子树诱导的子图，并在 x 的粘合集上具有某个特定的“签名”；可能的签名数量为 $2^{\mathcal{O}(k\ell^2)}$。

考察 [16] 中给出的证明，不难看出，这种动态规划可以理解为一种非确定性树分解自动机。因此，根据 Bodlaender 和 Kloks 的工作，我们可以立即推出以下结论。

引理 A.3. 对于每一对整数 $k \leqslant \ell$，都存在一个宽度为 ℓ 的非确定性树分解自动机$\mathcal{BK}_{k,\,\ell}$，具有以下性质：对于任意图 G 及其宽度至多为 ℓ 的二叉标注树分解 (T、袋、边) $\mathcal{BK}_{k,\,\ell}$，当且仅当 G 的树宽至多为 k 时，该自动机接受 (T、袋、边)。$\mathcal{BK}_{k,\,\ell}$ 的状态空间规模为 $2^{\mathcal{O}(k\ell^2)}$，可在时间 $2^{\mathcal{O}(k\ell^2)}$内计算。$\mathcal{BK}_{k,\,\ell}$ 的求值时间同样为 $2^{\mathcal{O}(k\ell^2)}$。

我们指出，由于树宽至多为 k 这一性质可以用 CMSO_2[5],引理 A.3 在求值时间界未明确给出的情况下，也可由 引理 A.2推出。原因是——

[5]例如，可以指出，给定的图不包含任何使树宽至多为 k 的禁用小图子式阻碍。已知此类阻碍的规模关于 k^5 有双指数上界，因此其数量至多为关于 k^5 [42] 的三重指数。

显式表述 引理 A.3 的原因，是为了在后续论证中更精确地跟踪求值时间。

闭包自动机。 最后，我们介绍用于计算树分解子树中小闭包的自动机。这些自动机将在以下内容的证明中使用：引理 4.23。我们首先需要一些定义。

设 G 是一个带边界图。我们称两个非边界顶点集合 Y， $Z \subseteq V(G) \setminus \partial G$是躯干图等价的，如果存在一个同构映射，将 $\mathsf{torso}_G(Y \cup \partial G)$ 映射到 $\mathsf{torso}_G(Z \cup \partial G)$，并固定 ∂G 中的每个顶点。注意，躯干图等价是一种等价关系；对每个整数 c，令 $\sim_{c,\, G}$ 为该等价关系在 $V(G) \setminus \partial G$ 的基数至多为 c 的子集上的限制。注意$\sim_{c,\, G}$ 至多有 $2^{\mathcal{O}((c+|\partial G|)^2)}$ 个等价类。

进一步假设 $\mathcal{T} = (T, \text{袋}, \text{边})$ 是 G 的一个带边界树分解。我们将第 3 节中定义的深度函数 $d_{\mathcal{T}}$ 推广到带边界树分解如下：令$d_{\mathcal{T}}\colon V(G) \setminus \partial G \to \mathbb{Z}_{\geqslant 0}$，使得 $d_{\mathcal{T}}(u)$ 是 T 中最上方且其袋包含 u 的节点的深度。特别地， $d_{\mathcal{T}}(u)$ 取决于顶点 u 和树分解 $(T, \text{袋}, \text{边})$。此外，对于一个顶点集合 $Y \subseteq V(G) \setminus \partial G$，我们定义 $d_{\mathcal{T}}(Y) = \sum_{u \in Y} d_{\mathcal{T}}(u)$。回顾一下，顶点 G 上继承自 Ω 的全序 $\preceq$ 允许按如下方式比较 $V(G) \setminus \partial G$ 的子集：对于 $Y, Y' \subseteq V(G) \setminus \partial G$，我们令 $Y \preceq_{\mathcal{T}} Y'$，如果

- $d_{\mathcal{T}}(Y) < d_{\mathcal{T}}(Y')$，或者
- $d_{\mathcal{T}}(Y) = d_{\mathcal{T}}(Y')$，并且 Y相对于 $\preceq$按字典序不大于 Y'。

对于 $V(G) \setminus \partial G$的一个非空子集族 $\mathcal{S}$，令 $\min_{\mathcal{T}} \mathcal{S}$ 为 $\mathcal{S}$中按 $\preceq_{\mathcal{T}}$排序最小的元素。由此，我们可以如下定义小躯干图代表元 c:

$$\begin{aligned}&\mathsf{reps}^c(G, (T, \mathtt{bag}, \mathtt{edges})) \coloneqq \\ &\{(d_{\mathcal{T}}(\min_{\mathcal{T}} \mathcal{K}), \mathtt{torso}_G(\min_{\mathcal{T}} \mathcal{K} \cup \partial G)) \colon \mathcal{K} \text{ is an equivalence class of } \sim_{c,G}\}.\end{aligned}$$

注意，$\sim_{c,G}$ 的每个等价类成员规模都相同，因此此处我们不对规模进行优化，尽管稍后证明引理 4.23时会用到规模。$\mathsf{reps}^c(G, (T, \text{袋}, \text{边}))$集合同时取决于带边界图 G 及其带边界树分解 $(T, \text{袋}, \text{边})$。此外，$\mathsf{reps}^c(G, (T, \text{袋}, \text{边}))$的基数等于 $\sim_{c,G}$的等价类数量，正如所述，该数量至多为 $2^{\mathcal{O}((c+|\partial G|)^2)}$。

我们将要使用的闭包自动机由以下陈述给出 nt.

引理 A.4. 对于每对整数 c, ℓ，存在一个树分解自动机 $\mathcal{R} = \mathcal{R}_{c,\, \ell}$，满足以下性质：对于任意图 G 及其宽度至多为 ℓ 的带注释的二叉树分解 $(T, \text{袋}, \text{边})$，$\mathcal{R}$ 在 $(T, \text{袋}, \text{边})$ 上的运行满足

$$\rho_{\mathcal{R}}(x) = \mathsf{reps}^c(G_x, (T_x, \mathtt{bag}_x, \mathtt{edges}_x)) \qquad \textit{for all } x \in V(T).$$

$\mathcal{R}$ 的求值时间为 $2^{\mathcal{O}((c+\ell)^2)}$。

证明。 对于状态空间 Q，我们取由形如 $(p,\ H)$ 的对所组成的所有集合，其中 $p \in \mathbb{N}$，且 H 是一个包含在 Ω 中、至多有 $c + \ell + 1$ 个顶点的图。 $\mathcal{R}$ 的终止状态与引理陈述无关，因此我们可以令 $F = \emptyset$。至于初始映射 ι，对于一个至多有 $\ell + 1$ 个顶点的带边界图 H，我们可以令 $\iota(H) =$代表元$^c(H,\ (T_0,\ \text{袋}_0, \text{边}_0))$，其中 $(T_0,\ \text{袋}_0, \text{边}_0)$ 是 H 的平凡单节点树分解，其中所有顶点

和边都放在根袋中。根据定义，显然可以直接在时间 $2^{\mathcal{O}((c+\ell)^2)}$ 内计算 $\iota(H)$。

还需定义转移映射 δ。为此，只需证明以下结论。设 $(T, \text{袋}, \text{边})$ 是图 G 的带边界树分解，且 x 是 G 中一个有子节点 y 和 z 的节点。已知

$$R_y := \mathsf{reps}^c(G_y, (T_y, \mathtt{bags}_y, \mathtt{edges}_y)), \qquad R_z := \mathsf{reps}^c(G_z, (T_z, \mathtt{bags}_z, \mathtt{edges}_z)),$$

以及 $\mathsf{bag}(x)$、$\mathsf{edges}(x)$ 和 x、y、z 的粘合集后，便可计算

$$R_x := \mathsf{reps}^c(G_x, (T_x, \mathtt{bags}_x, \mathtt{edges}_x))$$

所需时间为 $2^{\mathcal{O}((c+\ell)^2)}$。严格来说，对于 x 只有一个子节点 y 的情况，我们也需要类似的论证；不过，只需在两个子节点的论证中考虑一个袋为空的虚拟的第二个子节点 z 即可得出该结论。因此，我们只讨论两个子节点的情况。

对于任意一对集合 $Y \subseteq \mathsf{cmp}(y)$ 和 $Z \subseteq \mathsf{cmp}(z)$，定义 $G_x(Y, Z)$ 为一个顶点集为 $Y \cup Z \cup \mathsf{bag}(x)$ 的图，其边集由以下图的边集的并集构成：

$$\mathtt{torso}_{G_y}(Y \cup \mathtt{adh}(y)), \qquad \mathtt{torso}_{G_z}(Z \cup \mathtt{adh}(z)), \qquad \text{and} \qquad (\mathtt{bag}(x), \mathtt{edges}(x)).$$

下面的断言显而易见。

断言 A。5. 对于任意三组集合 $X \subseteq \text{袋}(x) \setminus \mathtt{adh}(x)$、$Y \subseteq \mathtt{cmp}(y)$ 以及 $Z \subseteq \mathtt{cmp}(z)$，我们有

$$\mathtt{torso}_{G_x}(X \cup Y \cup Z \cup \mathtt{adh}(x)) = \mathtt{torso}_{G_x(Y,Z)}(X \cup Y \cup Z \cup \mathtt{adh}(x)).$$

为了计算 R_x，我们首先按如下方式构造候选项集合族 C。考虑每一对对 $(p_y, H_y) \in R_y$ 和 $(p_z, H_z) \in R_z$，以及每个 $X \subseteq \text{袋}(x) \setminus \mathsf{adh}(x)$。令 $Y = V(H_y) \setminus \mathsf{adh}(y)$，并令 $Z = V(H_z) \setminus \mathsf{adh}(z)$；注意，图 $G_x(Y, Z)$ 是图 H_y、H_z 和 (袋(x),边(x)) 的并集。如果 $|X \cup Y \cup Z| \leqslant c$，那么我们向 C 添加这一对

$$(p_y + p_z + |Y| + |Z|, \mathtt{torso}_{G_x(Y,Z)}(X \cup Y \cup Z \cup \mathtt{adh}(x))).$$

否则，如果 $|X \cup Y \cup Z| > c$，则不向 C 添加任何对。添加到 C 的对的第一个坐标等于 $d_{T_x}(X \cup Y \cup Z)$，因为 $X \subseteq \text{袋}(x)$ 且 $Y, Z \subseteq V(G_x) \setminus \text{袋}(x)$。根据断言 A.5，添加到 C 的对的第二个坐标等于躯干图$G_x($ $X \cup Y \cup Z \cup \mathsf{adh}(x))$。

此外，我们有

$$|C| \leqslant |R_y| \cdot |R_z| \cdot 2^{\ell+1} \leqslant 2^{\mathcal{O}((c+\ell)^2)},$$

并且 C 可以在时间 $2^{\mathcal{O}((c+\ell)^2)}$内计算。

接下来，按如下方式筛选候选项。只要在 C 中存在不同的对 (p, H) 和 (p', H')，使得 H 与 H' 通过一个固定 $\mathsf{adh}(x)$ 的同构映射而同构，我们就删除第一坐标较大的那一对；如果两对的第一坐标相同，则删除第二坐标的顶点集在 $\preceq$中较大的那一对。显然，可以在时间 $2^{\mathcal{O}((c+\ell)^2)}$内穷举地执行这一筛选过程。

因此，经过筛选后，C 中各对的第二个坐标在$\sim_{c,\ G_x}$中两两不等价。通过一个简单的交换论证，可以直接看出，筛选后 R_x 等于 C。这就给出了转移函数 δ 的定义以及计算该函数的算法。

□

A.3 自动机运行的动态维护

定义了我们将要使用的自动机后，接下来说明如何在前缀重建更新下有效地维护其运行。

引理 A.6. 固定 $\ell \in \mathbb{N}$ 以及一个宽度为 ℓ、求值时间为 τ 的树分解自动机 $\mathcal{A} = (Q, F, \iota, \delta)$。那么存在一个开销为 τ 的 ℓ-前缀重建数据结构，它还实现了以下操作：

- 查询(x)：给定 T 的一个节点 x，返回 $\rho_{\mathcal{A}}(x)$。最坏情况时间为 $\mathcal{O}(1)$。

证明。 在任何时刻，数据结构都存储分解 $(T, 袋, 边)$，其中每个节点 x 都配有一个指向其父节点的指针、一对指向其子节点的指针，以及自动机 $\mathcal{A}$ 在 $(T, 袋, 边)$ 上的运行中对应的状态$\rho_{\mathcal{A}}(x)$。因此，可以按要求在常数时间内回答查询。

初始化时，我们只需自底向上计算 $\mathcal{A}$ 在 $(T, 袋, 边)$ 上的运行：根据初始化映射计算叶节点的状态，而内部节点的状态则根据转移映射自底向上计算。这需要每个节点耗时 τ，因此总计为 $\mathcal{O}(\tau \cdot |V(T)|)$。

要应用前缀重建更新 $u = (T_{\text{pref}}, T'前缀T^\star, 袋^\star, 边^\star, \pi)$，只需构建树 T'_{pref}，并根据 π 将 T_{pref} 在 T 中的所有附属节点重新连接起来（每个附属节点只需更改一个指针），即可轻松地在 $\ell^{\mathcal{O}(1)} \cdot |u|$ 时间内重建表示该分解的指针结构。请注意，对于重新连接的子树，在 $\mathcal{A}$ 上的运行信息无需更改；但对于 T'的前缀，其附属节点的粘合集可能发生变化，因此可能需要调整这些节点的运行信息。因此，只需计算新分解 $(T', 袋', 边')$ 中，运行 $\mathcal{A}$ 时与 T'前缀$\cup$ App(T'前缀) 的节点相关联的状态。为此，可以自底向上方式处理 T'前缀$\cup$App(T'前缀)，并通过初始化映射 ι（对于 T'前缀中属于 T'叶节点的节点）或转移映射 δ（对于 T'前缀中的其他节点）计算每个后续状态，总计耗时 $\mathcal{O}(\tau \cdot |T'$前缀$\cup$App($T'$前缀)$|) \leqslant \mathcal{O}(\tau \cdot |u|)$。 □

现在，引理 A.6 结合引理 A.2 立即推出引理 3.8。类似地，将 ℓ前缀重建数据结构应用于引理 3.5，即可立即得到一个实现其中前三种操作的结构。具体而言，将引理 A.6 应用于引理 A.1给出的自动机即可。下面我们补完引理 3.5的证明。

引理 3.5 的证明。根据以上讨论，只需实现一个开销为 $\mathcal{O}(1)$ 的 ℓ-前缀重建数据结构，用以实现操作 top(v)：给定顶点 $v \in V(G)$，返回满足 $v \in$ bag(t) 的唯一最高节点 t（属于 T）。考虑一次前缀重建更新，它将 $(T, \text{bag}, \text{edges})$ 改为 $(T', \text{bag}', \text{edges}')$。注意，只有当 $v \in \text{bags}_T(T_{\text{pref}} \cup \text{App}(T_{\text{pref}}))$ 时，前缀重建更新才会改变 v 出现的最高节点；特别是在这种情况下，$(T', \text{bag}', \text{edges}')$ 中 v 出现的最高节点将位于 $T'\text{pref} \cup \text{App}(T'\text{pref})$ 中。$|T_{\text{pref}} \cup \text{App}(T_{\text{pref}})|$ 和 $|T'\text{pref} \cup \text{App}(T'\text{pref})|$ 都是 $|u|$ 的线性函数，因此我们只需显式维护映射 top(v)，对 $v \in \text{bags}_T(T_{\text{pref}} \cup \text{App}(T_{\text{pref}}))$ 中的所有顶点重新计算即可。 □

A.4 闭包与阻塞

现在我们可以给出引理 4.23的证明。这需要更多工作：除了动态维护相关信息外，我们还需要实现提取闭包和阻塞集的方法。

引理 *4.23*的证明。为证明该引理，我们固定以下两个自动机：

- $\mathcal{R} = \mathcal{R}_{c,\ \ell}$ 是参数 c和 ℓ所对应的闭包自动机，由 引理 A.4给出。
- $\mathcal{BK} = \mathcal{BK}_{2k+1,\ c+\ell}$ 是参数 $2k+1$ 和 $c+\ell$所对应的 Bodlaender-Kloks 自动机，由 引理 A.3给出。

设 $\mathcal{BK} = (Q,\ F,\ \iota,\ \delta)$。回忆一下，$\mathcal{BK}$ 是非确定性的，$|Q| \leqslant 2^{\mathcal{O}(k(c+\ell)^2)}$，$Q$ 可在时间 $2^{\mathcal{O}(k(c+\ell)^2)}$内计算，并且对于相关对象，可以在时间 $2^{\mathcal{O}(k(c+\ell)^2)}$内判定其是否属于 F、ι或 δ。

我们的数据结构仅由引理 A.6 所给出的用于自动机 $\mathcal{R}$ 的数据结构构成。因此，初始化时间为 $2^{\mathcal{O}((c+\ell)^2)} \cdot |V(T)|$，更新时间为 $2^{\mathcal{O}((c+\ell)^2)} \cdot |u|$，符合要求。剩下的工作是实现方法 query($T^{\frown}_{\mathrm{pref}}$)。为此，我们可以假设，存储的带注释树分解 $(T,$ 袋，边) 已标记了自动机 $\mathcal{R}$ 在 $(T,$ 袋，边) 上的运行 $\rho_{\mathcal{R}}$。这意味着，对于每个 $x \in V(T)$，我们都可以访问 reps$^c(T_x,$ 袋$_x,$ 边$_x)$。

考虑查询 query(T_{pref})。我们将回答此查询分为两个步骤。第一步，我们计算 bags(T_{pref}) 的一个 $d_{\mathcal{T}}$-最小的 c-小闭包 k X，以及躯干图 torso$_G(X)$。这需要时间 $2^{\mathcal{O}(k(c+\ell)^2)} \cdot |T_{\mathrm{pref}}|$。第二步，我们求出阻塞集 Blockages($T_{\mathrm{pref}}, X$)。这需要时间$2^{\mathcal{O}((c+\ell)^2)} \cdot$|探索 Exploration($T_{\mathrm{pref}}, X$)|。

步骤 1：寻找闭包及其躯干图。 令 A 为 T_{pref} 的附属子树集合。对于 $a \in A$，令

$$R(a) := \mathsf{reps}^c(T_a, \mathsf{bag}_a, \mathsf{edges}_a);$$

回忆一下，$R(a)$ 与 a 一同存储在数据结构中。令 Λ 为所有映射 λ 的集合，其中定义域为 A，且对于所有 a，$\lambda(a) \in R(a)$。对于 $\lambda \in \Lambda$ 和 $a \in A$，令 $d^\lambda(a)$ 和 $H^\lambda(a)$ 分别为 $\lambda(a)$ 的第一和第二坐标，并令 $s^\lambda(a) = |V(H^\lambda(a)) \setminus \mathsf{adh}(a)|$。对于 $x \in V(T)$，我们用 $d_T(x)$ 表示 x 在 $(T, $袋$,$边$)$ 中的深度。对于 $\lambda \in \Lambda$，我们定义

$$\begin{aligned} H^\lambda &:= \left(\mathsf{bags}(T_{\mathrm{pref}}) \cup \bigcup_{a \in A} V(H^\lambda(a)), \mathsf{edges}(T_{\mathrm{pref}}) \cup \bigcup_{a \in A} E(H^\lambda(a)) \right), \\ s^\lambda &:= \sum_{a \in A} s^\lambda(a), \\ d^\lambda &:= \sum_{a \in A} \left(d^\lambda(a) + d_T(a) \cdot s^\lambda(a) \right). \end{aligned}$$

注意，根据 reps^c 的定义，我们有

- $H^\lambda = \mathtt{torso}_G(V(H^\lambda))$,
- $s^\lambda = |V(H^\lambda)| - |\mathsf{bags}(T_{\mathrm{pref}})|$, and
- $d^\lambda = d_{\mathcal{T}}(V(H^\lambda)) - d_{\mathcal{T}}(\mathsf{bags}(T_{\mathrm{pref}}))$

对所有 $\lambda \in \Lambda$均成立。

此外，对于每个 $a \in A$，集合 $R(a)$ 包含通过在 cmp(a) 中至多选取 c 个顶点所能得到的所有非同构躯干图；对于每种可能的躯干图 $R(a)$，其中都存储了总深度最小的一个实现。这立即推出以下结论。

命题 A.7。 设 $\lambda \in \Lambda$ 满足 H^λ 的树宽至多为 $2k+1$，且在所有满足此条件的映射 λ 中，s^λ 最小，并且在这些映射中 d^λ 最小。则 $V(H^\lambda)$ 是 $\mathsf{bags}(T_{\mathrm{pref}})$ 的一个 $d_{\mathcal{T}}$-最小 c-小 k-闭包。此外，如果不存在上述 λ，则不存在 $\mathsf{bags}(T_{\mathrm{pref}})$ 的 c-小 k-闭包。

因此，根据命题 A.7，我们只需找到 $\lambda \in \Lambda$，使其首先最小化 s^λ，其次最小化 d^λ，且 H^λ 的树宽至多为 k；或者得出不存在这样的 λ。事实上，这样我们就可以将 $X := V(H^\lambda)$ 和躯干图 $G[X] = H^\lambda$ 作为查询的输出。注意，一旦找到上述映射 λ，就可以根据定义直接在时间 $(\ell+c)^{\mathcal{O}(1)} \cdot |T_{\mathrm{pref}}|$ 内轻松构造 H^λ。直观地说，为了找到合适的 λ，我们针对不同的 λ，分析 Bodlaender-Kloks 自动机 $\mathcal{BK}$ 在从 T_{pref} 继承的 H^λ 的自然树分解上的可能运行。

令 $S = T_{\mathrm{pref}} \cup A$。对于 $x \in S$，令 S_x 为 S 中由 x 和 $A_x = S_x \cap A$ 的后代组成的子集，并令 Λ_x 的定义与 Λ 相同，只是定义域改为 A_x，而非 A。对于 $x \in V(T)$，令 $T_{\mathrm{pref},x} \subseteq T_{\mathrm{pref}}$ 由 T_{pref} 中属于 x 后代的节点组成。对于 $x, y \in V(T)$，令 $d_T(x,y)$ 表示它们在 T 中的距离。此外，对于 $\lambda \in \Lambda_x$，定义

$$\begin{aligned} H_x^\lambda &:= \left(\mathtt{bags}(T_{\mathrm{pref},x}) \cup \bigcup_{a \in A_x} V(H^\lambda(a)), \mathtt{edges}(T_{\mathrm{pref},x}) \cup \bigcup_{a \in A_x} E(H^\lambda(a)) \setminus \binom{\mathtt{adh}(x)}{2} \right), \\ s_x^\lambda &:= \sum_{a \in A_x} s^\lambda(a), \\ d_x^\lambda &:= \sum_{a \in A_x} \left(d^\lambda(a) + d_T(x,a) \cdot s^\lambda(a) \right). \end{aligned}$$

我们将 H_x^λ 视为边界为 $\mathtt{adh}(x)$ 的带边界图。于是，H_x^λ 自然继承自 $(T, \mathtt{bag}, \mathtt{edges})$，具有带边界的树分解 $(T_x^\lambda, \mathtt{bag}_x^\lambda, \mathtt{edges}_x^\lambda)$，如下所示：

- $T_x^\lambda = T[S_x]$；
- $\mathtt{bag}_x^\lambda(y) = \mathtt{bag}(y)$ 对所有 $y \in T_{\mathrm{pref}\,x}$ 均成立，并且 $\mathtt{bag}_x^\lambda(a) = V(H^\lambda(a))$ 对所有 $a \in A_x$ 均成立；
- 边$_x^\lambda(y) = $ 边$(y) \cup \bigcup_{a \in A_y} E(H^\lambda(a)) \cap \left(\binom{\mathtt{bag}(y)}{2} \setminus \binom{\mathtt{adh}(a)}{2} \right)$ 对所有 $y \in T_{\mathrm{pref}\,x}$，并且 边$_x^\lambda(a) = E(H^\lambda(a)) \setminus \binom{\mathtt{adh}(a)}{2}$ 对所有 $a \in A_x$。

注意，这个树分解的宽度至多为 $c+\ell$。

令 $\overline{\mathbb{Z}}$ 表示非负整数与 $+\infty$ 组成的集合。我们在 $\mathbb{Z} \times \mathbb{Z}$ 上对对采用全序：先比较第一个元素；若相等，再比较第二个元素。自底向上地，对每个 $x \in S$，我们计算映射 $\zeta_x \colon Q \times 2^{\binom{\mathtt{adh}(x)}{2}} \to \overline{\mathbb{Z}} \times \overline{\mathbb{Z}}$，定义如下：对于 $q \in Q$ 和 $W \subseteq \binom{\mathtt{adh}(x)}{2}$，$\zeta_x(q, W)$ 是满足下列条件的 $\lambda \in \Lambda_x$ 中 $(s_x^\lambda, d_x^\lambda)$ 的最小值

- $\mathcal{BK}$ 在 $(T_x^\lambda, 袋_x^\lambda, 边_x^\lambda)$ 上有一次运行，其中 x 被标记为 q；并且
- $\bigcup_{a \in A_x} E(H^\lambda(a)) \cap \binom{\mathtt{adh}(x)}{2} = W$.

如果不存在上述的 λ，我们就令 $\zeta_x(q, W) = (+\infty, +\infty)$。

现在我们说明，映射 ζ_x 可以用自底向上方式计算。这可由以下规则得出，其正确性显而易见。

- 对于每个 $a \in A$，$\zeta_a(q, W)$ 是满足以下条件的最小对 (s, d)：存在 $(d, H) \in R(a)$，具有以下性质：$\left(H - \binom{\mathtt{adh}(a)}{2}, q\right) \in \iota$, $E(H) \cap \binom{\mathtt{adh}(a)}{2} = W$，且 $|V(H) \setminus \mathtt{adh}(a)| = s$。
- 对于每个没有子节点的 $x \in T_{\mathrm{pref}}$，若满足 G_x, q，则 $\zeta_x(q, W) = (0,0)$；否则，$\in \iota$ 且 $W = \emptyset$，并且 $\zeta_x(q, W) = (+\infty, +\infty)$。

- 对于每个有一个子节点的 $x \in T_{\text{pref}}$，$\zeta_x(q, W)$ 是满足以下条件的最小值 (s, d)：存在 $q' \in Q$ 和 $W' \subseteq \binom{\mathtt{adh}(y)}{2}$，使得 $(s', d') = \zeta_y(q', W')$，且

$$\left(\left(\mathtt{bag}(x), \mathtt{adh}(x), \mathtt{adh}(y), \emptyset, \mathtt{edges}(x) \cup W' \setminus \binom{\mathtt{adh}(x)}{2}, q', \bot\right), q\right) \in \delta,$$
$$W = W' \cap \binom{\mathtt{adh}(x)}{2},$$
$$s = s', \text{ and}$$
$$d = d' + s'.$$

  如果不存在如上所述的 q', q'', W'，那么 $\zeta_x(q, W) = +\infty$。

- 对于每个有两个子节点 y 和 z 的 $x \in T_{\text{pref}}$，$\zeta_x(q, W)$ 是满足以下条件的最小 (s, d)：存在 $q', q'' \in Q$、$W' \subseteq \binom{\mathtt{adh}(y)}{2}$ 和 $W'' \subseteq \binom{\mathtt{adh}(z)}{2}$，使得 $(s', d') = \zeta_y(q', W')$ 且 $(s'', d'') = \zeta_z(q'', W'')$

$$\left(\left(\mathtt{bag}(x), \mathtt{adh}(x), \mathtt{adh}(y), \mathtt{adh}(z), \mathtt{edges}(x) \cup W' \cup W'' \setminus \binom{\mathtt{adh}(x)}{2}, q', q''\right), q\right) \in \delta,$$
$$W = (W' \cup W'') \cap \binom{\mathtt{adh}(x)}{2},$$
$$s = s' + s'', \text{ and}$$
$$d = d' + d'' + s' + s''.$$

  如果不存在上述的 q', q'', W', W''，那么 $\zeta_x(q, W) = +\infty$。

根据上述规则，所有针对 $x \in S$ 的映射 ζ_x 都可以在总计 $2^{\mathcal{O}(k(\ell+c)^2)} \cdot |T_{\text{pref}}|$ 时间内计算出来，因为 $|Q| \leqslant 2^{\mathcal{O}(k(\ell+c)^2)}$，且 $\mathcal{BK}$ 的求值时间为 $2^{\mathcal{O}(k(\ell+c)^2)}$。

根据 $\mathcal{BK}$ 的性质（见 引理 A.3），在所有使 H^λ 的树宽至多为 k 的 $\lambda \in \Lambda$ 中，最小的 (s^λ, d^λ) 等于 $\min_{q \in F} \zeta_r(q, \emptyset)$，其中 r 是 T 的根。已知 ζ_r 后，可以在 $2^{\mathcal{O}(k(\ell+c)^2)}$ 时间内计算出后一个最小值。最后，要找到实现该最小值的 $\lambda \in \Lambda$，只需按标准方式回溯动态规划。也就是说，计算映射 ζ_x 时，对于每个算出的 $\zeta_x(q, W)$ 值，我们都会记下该值的求得方式。找到使 $\zeta_r(q, \emptyset)$ 最小的 $q \in F$ 后，我们以自顶向下的方式，沿着 S 递归回溯 $\zeta_r(q, \emptyset)$ 值的求得过程，直至 A 的节点中计算出的值；这些值的求得方式为我们提供映射 λ。至此，闭包 $X = V(H^\lambda)$ 及其躯干图 H^λ 的构造完成。

步骤 2：寻找阻塞集。构造出 $X = V(H^\lambda)$ 以及其躯干图 $_G(X) = H^\lambda$ 后，我们接着寻找集合 Blockages(T_{pref}, X)。我们可以假设 X 的每个顶点都已标记为属于 X（这可以在计算 X 后用 $\mathcal{O}(|X|) \leqslant (c+\ell)^{\mathcal{O}(1)} \cdot |T_{\text{pref}}|$ 时间完成），因此可以在常数时间内检查给定顶点是否属于 X。

首先，我们注意到，对于每个节点 $x \in V(T) \setminus T_{\text{pref}}$，都可以高效地找出 X 在以 x 为根的子树中的行为信息。记 $X_x := X \cap \mathsf{cmp}(x)$。

命题 A.8. 给定一个节点 $x \in V(T) \setminus T_{\text{pref}}$，可以在时间 $2^{\mathcal{O}((c+\ell)^2)}$ 内计算出 X_x 和躯干图 $G_x(X_x \cup \mathtt{adh}(x))$。

证明。对于每个 $(d, H) \in$ 代表元$^c(T_x, \text{袋}_x, \text{边}_x)$，若满足 $V(H) \setminus \mathsf{adh}(x) \subseteq X$，则称 H 为一个候选项。注意，只需逐一检查 $\mathsf{reps}^c(T_x, \mathsf{bag}_x, \mathsf{edges}_x)$ 中的所有元素，即可在时间 $(c+\ell)^{\mathcal{O}(1)} \cdot |\mathsf{reps}^c(T_x, \mathsf{bag}_x, \mathsf{edges}_x)| \leqslant 2^{\mathcal{O}((c+\ell)^2)}$ 内找出所有候选项。现在，简单的交换论证表明，$X_x \cup \mathsf{adh}(x)$ 等于最大的候选项（按顶点数量计），而 $\mathsf{torso}_{G_x}(X_x \cup \mathsf{adh}(x))$ 等于该候选项。 □

出于技术目的，我们需要建立一个简单的数据结构来检查 H^λ 中的邻接关系。这个思路基于退化度的概念，可以说是众所周知的结果。

断言 A.9. 在$(c+\ell)^{\mathcal{O}(1)} \cdot |T_{\text{pref}}|$ 时间内，可以建立一个数据结构，对于给定的顶点 u、$v \in X$，能够在 H^λ 时间 $\mathcal{O}(\ell)$内判定 u 和 v 是否相邻。

证明。 H^λ 的树宽至多为 $2k+1 = \mathcal{O}(\ell)$，这意味着我们可以在 $\mathcal{O}(k \cdot |X|)$时间内计算出 X 上的一个全序 $\preceq$，使得每个顶点 $u \in X$ 在 $\preceq$中排在 u 之前的邻居至多有 $2k+1$ 个。因此，对于 X 的每个顶点 u，我们构造一个列表 $L(u)$，其中包含 u 在 $\preceq$中排在 u 之前的所有邻居。通过逐一检查 H^λ 的边即可构造这些列表；对于每条下一条边 uv，根据 u 或 v 在 $\preceq$中的先后顺序，将 u 追加到 $L(v)$，或将 v 追加到 $L(u)$。然后，通过检查 u 是否出现在 $L(v)$中，或 v 是否出现在 $L(u)$中，可以在 $\mathcal{O}(k)$时间内判定给定的 u 和 v 是否相邻。 □

接下来，对于节点 $x \in V(T)$，我们将剖面$(x) \subseteq \binom{\mathsf{bag}(x)}{2}$ 定义为由所有对$\{u, v\} \subseteq$ 袋(x) 组成的集合，使得在 G 中存在一条连接 u 与 v 且与 X内部不相交的路径。（注意，此定义涉及 u 和 v 既属于也不属于 X的情况。）注意，如果$x \in T_{\text{pref}}$，则有袋$(x) \subseteq X$，且剖面(x)由躯干图$_G(X) = H^\lambda$ 中两个端点都在袋(x)内的所有边组成。因此，对于这样的 x，我们可以使用断言 A.9 的数据结构在时间 $\ell^{\mathcal{O}(1)}$ 内计算剖面(x)。

Next，我们展示了已知父节点的剖面后，可以计算子节点的剖面。 d.

命题 A.10。假设 x 是 T和 $y \notin T_{\text{pref}}$中的 y 的父节点。那么给定剖面(x)，可以在时间 $2^{\mathcal{O}((c+\ell)^2)}$内计算剖面$(y)$。

证明。令 J 为顶点集 $\mathsf{bag}(y)$ 上的图，其边集是以下各项的并集：

- $\mathsf{edges}(y)$ 中的所有边；
- $\mathsf{profile}(x)$ 中两端点都在 $\mathsf{adh}(x)$ 内的所有边；以及
- 对于每个子节点 z，$\mathsf{torso}_{G_z}(X_z \cup \mathsf{adh}(z))$ 中两端点都在 $\mathsf{adh}(z)$ 内的所有边。

注意，利用命题 A.8，可以在时间 $2^{\mathcal{O}((c+\ell)^2)}$ 内构造 J。现在很容易看出，$\mathsf{profile}(y)$ 由 $\mathsf{bag}(y)$ 中所有这样的对 $\{u, v\} \subseteq$ 组成：在 J中存在一条路径连接 u和 v，且该路径与 $X \cap \mathsf{bag}(y)$ 内部不相交。根据这一观察，由于 J至多有 $\ell+1$ 个顶点，现在可以在时间 $\ell^{\mathcal{O}(1)}$ 内构造 $\mathsf{profile}(y)$。 □

在确立命题 A.10 后，我们终于可以描述寻找阻塞的过程。该过程逐一检查附属子树 $a \in A$，并在检查 a 时找出所有作为 a 后代的阻塞。为此，我们从 a 开始，在 T_a 中进行深度优先搜索。搜索过程中，我们始终将当前处理的节点 $x \in V$（T_a）及其剖面(x)一并存储。最初，可以利用命题 A.10 计算剖面(a)；如前所述，属于 T_{pref} 的 a 的父节点的剖面可以在时间 $\ell^{\mathcal{O}(1)}$ 内计算。当搜索从父节点 x 进入节点 y 时，可以再次利用命题 A.10，根据剖面(x)计算剖面(y)。注意，已知剖面(x)后，可以在时间 $(c+\ell)^{\mathcal{O}(1)}$ 内确定 x 是否为阻塞：

- 如果 $\text{bag}(x) \subseteq X$ 且剖面$(x) = \left(\text{bag}(2^x)\right)$，那么 x 是团阻塞。
- 若袋$(x) \cap X \neq \emptyset$ 且存在 $u \in$ 袋$(x) \setminus X$ 使得对于所有$v \in$ 袋$(x) \setminus \{u\}$，都有 $\{u, v\} \in$ 剖面(x)，则 x 是一个分量阻塞。

因此，如果 x 是阻塞集，我们就输出 x，并不再继续搜索。否则，如果 x不是阻塞集，搜索就递归到 x的子节点。

显然，搜索访问的节点总数不超过 $|$探索$(T_{\text{pref}}, X)|$，且每个节点所需时间为 $2^{\mathcal{O}((c+\ell)^2)}$。因此，算法输出所有阻塞集的总时间为 $2^{\mathcal{O}((c+\ell)^2)}|$探索$(T_{\text{pref}}, X)|$。

□

B 定理 1.1 的证明定理 1.1

本节将完成定理 1.1的证明。

我们先简要概述证明思路。回顾一下，根据引理 3.9，存在一种数据结构，它针对初始为空的动态图 G，在边插入和删除操作更新下，维护一个带注释的树分解$(T,$ 袋，边$)$，其 G 宽度至多为 $6k + 5$，并保证 G 的树宽始终不超过 k。现在，我们逐步改进该数据结构，使其更能应对导致$\text{tw}(G)$ 超过 k的查询，然后增加对任意CMSO_2 性质的测试支持：

- 第一项改进（引理 B.1）使数据结构能够检测出会使 G 的树宽增加到超过 k的更新，从而使我们能够拒绝这些更新（即拒绝执行此类更新，并保持数据结构的状态不变）。具体做法是：（a）维护一个动态 G'，使其在每次更新后都等于 G，并额外维护一个引理 3.9 中的数据结构实例，用于维护 G' 的树分解，其宽度至多为 $6k + 11$，并以 $\text{tw}(G') \leqslant k+1$ 为前提；（b）同时维护 Bodlaender–Kloks 自动机（引理 A.3），给定 G' 的宽度至多为 $6k + 11$的增广树分解后，用于验证 G 的树宽是否至多为 k；（c）首先将每个更新应用于 G'，使用 Bodlaender–Kloks 自动机验证更新后是否有 $\text{tw}(G') \leqslant k$，并根据是否满足这一条件来接受或拒绝该更新。

- 下一项改进（引理 B.3）使数据结构能够实际处理会使 $\text{tw}(G)$ 超过 k的更新。在任何时刻，数据结构都会维护以下信息：是否$\text{tw}(G) \leqslant k$，以及 G 最新快照的一个带注释的树分解，其树宽至多为 k。这是对 Eppstein 等人提出的延迟会破坏不变量的插入操作 这一技术的相当标准的应用。 [29]这项改进解决了定理 1.1的前一项断言。

- 为了证明定理的后一项断言（即维护任意 CMSO_2性质），我们将引理 B.3 中的数据结构与引理 3.8中的前缀重建数据结构结合使用。

这三个步骤都应视为标准且相当直接；例如，前两个步骤用于对 Majewski 等人的数据结构进行类似的改进，以维护反馈顶点数有界的图的 CMSO_2 性质。 [43]

引理 B.1. 存在一种数据结构，对于在初始化时固定的整数 $k \in \mathbb{N}$，以及通过边插入和删除进行更新的动态图 G，能够维护一个带注释的树分解

$(T,$ 袋，边$)$ 关于 G 的分解，其宽度至多为 $6k+5$，采用前缀重建更新，且仅当更新后的图的树宽至多为 k 时才接受这些更新。更准确地说，在任意时刻，该图的树宽至多为 k，且该数据结构包含一个关于 G 的带注释树分解，其宽度至多为 $6k+5$。该数据结构可在时间 $2^{k^{\mathcal{O}(1)}} \cdot n$ 内对 k 和一个无边的 n 顶点图 G 进行初始化，之后每次更新：

- 需要摊销时间 $2^{k^{\mathcal{O}}(1)\cdot\sqrt{\log n \log\log n}}$；
- 如果更新后的图的树宽大于 k，则拒绝该更新，并返回“树宽过大”；
- 否则，接受该更新，并由数据结构返回一系列前缀重建更新，用于从$(T,$ 袋，边$)$得到新图的带注释的树分解。

证明。 我们通过设置三个现有数据结构的实例来实现所需的数据结构：

- $\mathbb{D}_k$：来自 引理 3.9 的数据结构，用于维护动态图的带注释的树分解 G_k，其宽度至多为 $6k+5$，并在满足以下承诺的条件下使用前缀重建更新：$\mathrm{tw}(G_k) \leqslant k$；
- $\mathbb{D}_{k+1}$：一个类似的数据结构，用于维护动态图 G_{k+1} 的带注释的树分解，其宽度至多为 $6k+11$，并在满足 $\mathsf{tw}(G_{k+1}) \leqslant k+1$这一承诺的条件下使用前缀重建更新；
- $\mathbb{BK}$：一个（$6k+11$）-前缀重建数据结构，开销为 $2^{\mathcal{O}(k^3)}$，用于维护带注释的树分解所维护的动态图的树宽是否至多为 k；结合引理 A.3 和A.6可知，这种数据结构确实存在。

所有数据结构都以一个包含 n 个顶点的无边图初始化。在对数据结构进行更新期间，我们维持以下不变量：三个数据结构都存储着同一个树宽至多为 k 的动态图的描述；此外，$\mathbb{D}_{k+1}$ 和 $\mathbb{BK}$ 存储的带注释的树分解完全相同：$\mathbb{D}_{k+1}$ 执行的每次前缀重建更新也会应用于 $\mathbb{BK}$。每次更新成功时，数据结构都会返回应用于 $\mathbb{D}_k$中存储的带注释的树分解的所有前缀重建更新。

剩下的问题是说明如何处理这些更新。

- 假设从图中删除了一条边 uv。注意，边删除不会增加所维护图的树宽，因此可以安全地将该删除操作转发给所有数据结构。
- 当要向图中添加一条边 uv 时，我们首先通过添加这条边来更新 $\mathbb{D}_{k+1}$。由于向图中添加一条边最多会使其树宽增加 1，$\mathbb{D}_{k+1}$ 将处理此更新。如果更新后，$\mathbb{BK}$ 确认所存储图的树宽仍至多为 k，我们就通过将 uv 也添加到由 $\mathbb{D}_k$存储的图中来接受此更新。

  另一方面，如果 $\mathbb{BK}$ 返回更新后图的树宽大于 k，那么我们就拒绝该更新：通过从存储的图中移除 uv 来更新 $\mathbb{D}_{k+1}$，并返回“树宽过大”。[6]

[6]请注意，尽管此情况下更新被拒绝，且 $\mathbb{D}_k$ 和 $\mathbb{D}_{k+1}$ 存储的图并未改变，但由 $\mathbb{D}_{k+1}$ 存储的带注释的树分解最终仍可能因该更新而被修改。

容易证明，这些更新保持了规定的不变量，并且对于数据结构的每次查询， $\mathbb{D}_k$ 和 $\mathbb{D}_{k+1}$ 都会被更新常数次，因此这些数据结构满足所需的时间复杂度界。由引理 3.9 还可知，在前若干次查询中，

$\mathbb{D}_{k+1}$ q $2^{k^{\mathcal{O}(1)}} \cdot n$ +执行的前 $2^{k^{\mathcal{O}(1)} \cdot \sqrt{\quad}}$ n $\mathbb{BK}$

缀重建更新总规模为$\log \log \log^n \cdot q$。由于 $\mathbb{D}_{k+1}$的每次更新也会应用于（一个开销为 $2^{\mathcal{O}(k^3)}$）的前缀重建数据结构，因此，数据结构最初的 q次更新由 $\mathbb{BK}$ 在时间 $2^{k^{\mathcal{O}(1)}} \cdot n + 2^{k^{\mathcal{O}(1)} \cdot \sqrt{\log n \log \log^n}} \cdot q$内处理。由此证明了该数据结构的摊销时间复杂度。

□

现在，我们增强引理 B.1 中的数据结构，使其能够接受使该图树宽增加的边插入 k。为此，我们采用前面提到的延迟破坏不变量的插入技术。这里，我们使用 Chen 等人 [20]提出的该技术表述。

设 U 是一个全集。若集合族 $\mathcal{F} \subseteq 2^U$ 满足：对于所有 $E \subseteq F$，只要 $F \in \mathcal{F}$，就有 $E \in \mathcal{F}$，则称其为向下封闭。假设 $\mathbb{F}$ 是一个数据结构，用于维护动态子集 $X \subseteq U$，并支持元素的插入和删除。我们称 $\mathbb{F}$:

- 强支持$\mathcal{F}$隶属关系，如果 $\mathbb{F}$ 支持布尔查询 member()，该查询会返回是否 r $X \in \mathcal{F}$;
- 弱支持$\mathcal{F}$隶属关系，如果 $\mathbb{F}$ 维护一个不变量，即 $X \in \mathcal{F}$，并拒绝会破坏该不变量的插入。

那么，以下陈述给出了一种将对 $\mathcal{F}$ 隶属关系的弱支持转化为强支持的方法:

引理 B.2 ([20], 引理 11.1)。 假设 U 是一个全集，并且我们可以访问定义在 U上的字典 $\mathbb{L}$。设 $\mathcal{F} \subseteq 2^U$ 向下封闭，并假设存在一个数据结构 $\mathbb{D}$，能够弱支持 $\mathcal{F}$隶属关系。

那么，存在一个数据结构 $\mathbb{D}'$，能够强支持 $\mathcal{F}$ 隶属关系，其中每次成员() 查询需要 $\mathcal{O}(1)$ 时间，每次更新需要摊销 $\mathcal{O}(1)$ 时间，并且摊销需要 $\mathcal{O}(1)$ 次对$\mathbb{L}$ 和 $\mathbb{D}$执行操作的调用。此外， $\mathbb{F}'$ 维护一个 $\mathbb{F}$ 的实例，并且每当成员() $=$ 为真时， $\mathbb{F}$ 存储的集合与 $\mathbb{F}'$存储的集合 $X \in \mathcal{F}$ 相同。

引理 B.2的最后一条陈述没有正式表述；不过，从 [20]中的证明可以直接得出这一点。

接下来，我们说明引理 B.2 如何应用于引理 B.1中的数据结构。

引理 B.3。 存在一种数据结构，对于一个在初始化时固定的整数 $k \in \mathbb{N}$，以及一个通过边插入和删除进行更新的动态图 G，能够维护:

- 一个宽度至多为 $6k+5$、针对最近快照的树宽至多为 k 的带注释的树分解$(T$，袋，边$)$;
- 关于$\mathrm{tw}(G) \leqslant k$的布尔信息。

该数据结构可在时间 $2^{k^{\mathcal{O}(1)}} \cdot n$ 内针对 k 和一个具有 n 个顶点的无边图 G 完成初始化，之后每次更新:

- 耗时为 $2^{k^{\mathcal{O}}{}_{(1)} \cdot \sqrt{\log n \log \log^n}}$;

- 如果更新后图的树宽大于 k，则返回“树宽过大”；否则，返回用于修改(T、袋、边)的前缀重建更新序列。

证明。沿用 Chen 等人的记号，定义全集 $U := \binom{V(G)}{2}$。此外，定义集合族 $\mathcal{F} \subseteq 2^U$ 如下：当且仅当图 H 在包含 $V(H) = V(G)$ 和 $E(H) = X$的情况下，树宽至多为 k时，$X \in \mathcal{F}$。显然，$\mathcal{F}$ 是向下封闭的。再令 $\mathbb{F}$ 为引理 B.1 的数据结构。注意，$\mathbb{F}$ 弱支持 $\mathcal{F}$：它处理边添加和删除，并且仅当更新后的图的树宽至多为 k时才接受这些更新。因此，由引理 B.2，存在一个强支持 $\mathcal{F}$的数据结构 $\mathbb{F}'$。此外，$\mathbb{F}'$ 维护 $\mathbb{F}$ 的一个实例；当所维护图的树宽至多为 k时，$\mathbb{F}$ 存储的图与 $\mathbb{F}'$相同。

现在，我们的数据结构按照上述方式维护 $\mathbb{F}$ 和 $\mathbb{F}'$ 的实例，以及自上次 G 的树宽不超过 k 以来，对 $\mathbb{F}$ 执行的前缀重建更新序列 s。每次更新都原样转发给 $\mathbb{F}'$；随后，$\mathbb{F}'$ 通过边插入和删除来更新 $\mathbb{F}$，使得 $\mathbb{F}$ 使用一系列前缀重建更新来重建其带注释的树分解。此类更新会追加到 s。如果在更新我们的数据结构后，$\mathbb{F}'$ 返回 member() = false，那么我们就返回“树宽过大”。否则，我们返回前缀重建更新序列 s，并在下一次查询之前清空 s。

不难证明，所描述的数据结构满足引理的所有要求。

□

定理 1.1 的证明现可直接给出：

定理 1.1 的证明。定理陈述的第一部分可由引理 B.3 直接得到。现在，假设给定一个 CMSO_2 公式 φ，我们需要验证当 $\mathrm{tw}(G) \leqslant k$ 时是否有 $G \models \varphi$。为此，我们调用引理 3.8，并实例化一个 $(6k+5)$-前缀重建数据结构 $\mathbb{M}$，其开销为 $\mathcal{O}_k,\ \varphi(1)$，且可在最坏情况时间 $\mathcal{O}_k,\ \varphi(1)$ 内查询 $G \models \varphi$。现在，每当引理 B.3 的数据结构返回 $\mathrm{tw}(G) \leqslant k$ 时，它都会返回一系列前缀重建更新，我们立即将其转发给 $\mathbb{M}$。然后，我们查询 $\mathbb{M}$，以验证是否有 $G \models \varphi$。

由引理 B.3 立即可知，在前 q 次查询中所有前缀重建更新的总规模至多为 $2^{k^{\mathcal{O}(1)}} \cdot n + 2^{k^{\mathcal{O}(1)} \cdot \sqrt{\log n \log\log n}} \cdot q$。因此，$\mathbb{M}$ 在时间 $\mathcal{O}_k,\ \varphi(2^{k^{\mathcal{O}(1)}} \cdot n + 2^{k^{\mathcal{O}(1)} \cdot \sqrt{\log n \log\log n}} \cdot q)$ 内处理前 q 次查询，这满足所设定的摊销时间复杂度界。

□