单调动态规划的动态维护及其应用

Monika Henzinger ！ 计算机科学学院，维也纳大学
StefanNeumann ！ 瑞典斯德哥尔摩，皇家理工学院
Harald Räcke ！ 慕尼黑工业大学，德国慕尼黑
Stefan Schmid ！ 柏林工业大学，德国；弗劳恩霍夫安全信息技术研究所，德国

arXiv:2301.01744v1 [cs.DS] 4 Jan 2023

摘要

动态规划（DP）是算法设计中的基本范式之一。然而，许多动态规划算法必须填充大型动态规划表，并以二维数组表示，导致运行时间和空间开销至少为二次。这促使人们针对动态规划满足 Monge 性质或全单调性等附加性质的特殊情形，开发更高效的算法。

本文考虑一种新条件：假设（除其他一些技术性假设外）动态规划表的行是单调的。在此假设下，我们提出一种新颖的数据结构，在静态情形下以近线性时间和空间计算 $(1+\epsilon)$-近似动态规划解；当动态规划表项动态变化时，则具有多对数更新时间。据我们所知，这一新条件与以往条件互不包含，并且是首个能够基于现有动态规划推导出动态算法的条件。我们不再使用二维数组存储动态规划表，而是用单调分段常数函数存储其各行。这样，只需 polylog$(n,\ W)$ 位，就能存储长度为 n、表项取值属于 $[0, W]$ 的动态规划表行；此外，还能以多对数时间对这些函数执行操作，例如 $(\min,+)$-卷积或舍入。

我们进一步介绍了该数据结构的若干应用。对于 k-平衡图划分 和同时源定位的双准则版本，我们得到了首个更新时间为次多项式的动态算法，以及首个仅需近线性时间和空间的静态算法。此外，我们还得到了目前最快的全动态背包算法。对于k-平衡划分，我们展示了如何将 Feldmann 和 Foschini（Algorithmica'15）提出的现有非单调动态规划单调化；对于同时源定位，我们通过考虑 Andreev、Garrod、Golovin、Maggs 和 Meyerson（TALG'09）所用动态规划函数的逆动态规划函数，得到了一个高效算法。我们在全动态背包问题上的结果改进了 Eberle、Megow、Nölke、Simon 和 Wiese（FSTTCS'21）近期提出的结果。

2012年ACM主题分类 计算理论 → 动态规划；计算理论 → 动态图算法；计算理论 → 装箱与覆盖问题

关键词和短语 动态规划、动态算法、数据结构

资助 *Monika Henzinger*：本项目获得了欧洲研究理事会（ERC）通过欧盟地平线2020研究与创新计划提供的资助（资助协议编号：101019564，"现代全动态数据结构的设计（MoDynStruct）"），以及奥地利科学基金（FWF）项目"反应式网络层的快速算法（ReactNet）"（P 33775-N）的资助，并获得*netidee SCIENCE*基金会的额外资助，2020—2024年。*Stefan Neumann*：这项研究由ERC高级资助项目REBOUND（834862）和EC H2020 RIA项目SoBigData++ （871042）资助。

Stefan Schmid：研究获奥地利科学基金（FWF）项目 I 5025-N（DELTA）资助，项目周期为 2020-2024。

2 单调动态规划的动态维护及其应用

目录

1 引言 4 2 维护单调动态规划表 8

3 全动态背包 **11**
3.1 单调函数卷积求解背包问题.11 3.2 动态维护小规模实例.14

4 技术概述 1 **7**
4.1 将 Feldmann 和 Foschini 的动态规划单调化.17 4.1.1 计算动态规划.19 4.2 对 Andreev 等人的动态规划求逆.22

A 附录结构 **26**

B 更多相关工作 **26**

C 预备知识 28
C.1 Räcke 树.28 C.2 良性 DP.29

D 平衡图划分 30
D.1 精确动态规划.31 D.1.1 动态规划定义.33 D.1.2 计算动态规划.33 D.2 近似动态规划.36 D.3 计算结果.40 D.4 扩展到一般图.43 D.5 扩展到动态设置.45

E 同时源定位 48
E.1 精确动态规划.49 E.1.1 动态规划定义.49 E.1.2 计算动态规划.51 E.2 近似动态规划.53 E.3 推广到一般图（定理 4 的证明）.54 E.4 推广到动态设置（定理 5 的证明）.54

F 重调整界 56
F.1 定理 34 的证明.57 F.2 定理 35 的证明.59

G 非单调函数与 ℓ_∞-项链对齐 **60**
G.1 含非单调性的分段常数函数.60 G.1.1 引理 40 的证明.62 G.2 ℓ_∞-项链对齐.64 G.2.1 静态算法.65 G.2.2 动态算法.68

H 省略的证明 **69**
H.1 引理 6 的证明.69 H.2 引理 7 的证明.70 H.3 定理 9 的证明.71 H.4 定理 10 的证明.72 H.5 Räcke 树的性质.72 H.6 引理 18 的证明.73 H.7 引理 19 的证明.73 H.8 引理 20 的证明.73 H.9 引理 21 的证明.73 H.10 引理 22 的证明.74

1 引言

动态规划（DP）是算法设计中的基本范式之一。在动态规划范式中，一个复杂问题被拆解为更简单的子问题，然后通过组合这些子问题的解来解决原问题。动态规划算法的一个缺点是，在实践中它们往往运行缓慢且占用大量内存：对于规模为 n 的输入，其运行时间通常为 $\Omega(n^2)$；而当动态规划表以二维数组存储时，所需空间也为 $\Omega(n^2)$。

这促使研究人员开发运行时间和空间均为近线性的高效动态规划算法。事实上，在动态规划表 $[2,12,19,22,31,35,45,48,49,64]$, 满足一系列广泛条件时，这类改进是可行的，例如 Monge 性质、全单调性、某些凸性和凹性性质，或 Knuth–Yao 四边形不等式；我们将在附录 B 中更详细地讨论这些性质。当这些性质成立时，通常无需计算整个动态规划表，而只需计算能够揭示最优解的 $O(n)$ 个动态规划表项。

然而，据我们所知，目前还没有发现任何能够导出高效动态 算法的动态规划性质，也就是说，没有能够在输入发生变化时高效执行更新操作的算法。从概念上看，这或许有些令人意外，因为许多动态算法会对输入进行层次化划分，并维护各个子问题的解；这与许多动态规划方案的推导方式十分相似。事实上，许多“手工设计”的算法（例如 $[26,38]$）都利用了这种概念上的相似性：它们从一个动态规划方案出发，再说明如何在输入变化时动态维护该方案。然而，这类算法往往相当复杂，其分析通常需要精巧的摊还分析方案。

因此，自然会提出这样一个问题：是否存在一种通用判据，只要满足该判据，就能保证针对给定的动态规划，在输入发生变化时高效地进行更新?

我们的贡献。 本文的主要贡献是提出一种通用判据，使得动态规划表中的所有条目都可以在因子 $1+\epsilon$ 的范围内得到近似。我们证明，若某个动态规划满足我们的判据（并采用适当的参数），则：

- 在动态设置下，我们可以用多对数更新时间维护整个动态规划表的 $(1+\epsilon)$-近似（见定理 10）。在静态情形下，我们可以在近线性时间和空间内计算动态规划表的 $(1+\epsilon)$-近似（见定理 9）。

我们的判据本质上断言，动态规划表的行 应当单调 ，且动态规划的依赖图应为有向无环图，其中可达节点集合较小，同时还需满足其他一些技术条件（形式化定义见定义 8）。我们的判据与 Monge 性质、全单调性或文献中的其他判据互不包含（更详细的讨论见附录 B ）。

为得到我们的结果，我们为满足我们判据的动态规划引入了一种新颖的数据结构。我们的数据结构基于这样一种思路：使用单调分段常数函数存储动态规划行。动态规划行的单调性使我们能够确保这些函数只包含很少的分段。随后，我们证明可以非常高效地对这类函数执行操作，运行时间仅取决于分段数。这一点至关重要，因为它使我们能够在 $\text{polylog}(W)$ 时间内计算出一整行 完整的$(1+\delta)$-近似动态规划，即使动态规划有 $\Omega(n)$ 列，前提是动态规划表项取自 $[0,W]$。请注意，如果 $W \le \text{poly}(n)$，那么计算整行的运行时间便会从 $\Omega(n)$ 降至 $\text{polylog}(n)$。此外，这还使我们能够存储每一行，只需$\text{polylog}(W)$ 空间，而不必将其存储在大小为 $\Omega(n)$ 的数组中。我们在第 2 节介绍我们的判据以及数据结构的细节。

作为我们数据结构的应用，我们为多个问题得到了新的静态和动态算法。我们针对 k-平衡划分、同时源定位以及全动态背包提出了新的算法。接下来，我们将详细介绍这些结果；更多相关工作请参见附录 B。

完全动态 0-1 背包问题的研究结果。 首先，我们为完全动态 0-1 背包问题提出了一种新算法。在该问题中，输入由一个背包容量 $B \in \mathbb{R}_+$和一组 n 物品组成，每件物品 $i \in [n]$ 都有一个权重 $w_i \in \mathbb{R}_+$ 和一个价格 $p_i \in [1, \infty)$。目标是在满足约束条件$\sum_{i \in I} w_i \leq B$的同时，找到一组物品 I，使 $\sum_{i \in I} p_i$ 最大化。在该问题的动态版本中，物品会被插入和删除。更具体地说，我们考虑以下更新操作：插入(p_i, w_i)，其中物品 i 的价格和权重分别设为 $p_i \in [1, \infty)$ 和 $w_i \in \mathbb{R}_+$，以及删除(i)，即从物品集合中移除物品 i。

我们的主要结果是一种动态 $(1+\epsilon)$-近似算法，其最坏情况更新时间为 $\epsilon^{-2} \cdot \log^2(nW) \cdot \text{polylog}(1/\epsilon, \log(nW))$, 其中 $W = \sum_i p_i$。我们的算法改进了 Eberle、Megow、Nölke、Simon 和 Wiese 最近提出的结果 [29]，他们同样维护了一个 $(1+\epsilon)$-近似解，更新时间为 $O(\epsilon^{-9} \log^4(nW))$。

▶**定理 1.** 设 $\epsilon > 0$。存在一种用于全动态背包问题的算法，可以维护一个$(1+\epsilon)$-近似解，其最坏情况更新时间为 $\frac{1}{\epsilon^2} \log^2(nW) \operatorname{polylog}\left(\frac{1}{\epsilon} \log(nW)\right)$。

我们还将证明，可以在 $O(|I|)$ 时间内返回维护的解 I，并且可以在 $O(1)$ 时间内查询给定物品 $i \in [n]$ 是否包含在 I 中。这与 [29]的查询时间相匹配。

为得到这一结果，我们首先将维护具有单调行的动态规划的数据结构作为简单应用，推导出一个稍慢一些的算法。然后，我们结合这一算法与其他思路，得到该定理（见第 3 节）。

由于我们的动态算法基于动态规划，每次更新后，解都有可能发生显著变化。不过，我们在附录（定理 34）中证明了一个下界，表明每个用于背包问题的动态 $(1+\epsilon)$-近似算法，要么必须在每次更新后对解做大量修改，要么必须存储许多（可能差异很大的）解，并在每次更新后从中切换。这意味着，无法以多对数更新时间维护单个显式解，因此我们的算法所具有的这一性质无法避免。

我们关于 k-平衡划分的结果。我们在技术上最具挑战性的结果是关于 k-平衡图划分的。在这个问题中，输入由一个整数 k 和一个无向加权图 $G = (V, E, \text{cap})$ 组成，该图有 n 个顶点，其中 $\text{cap}: E \to W_\infty$ 是定义在边上的权重函数，权重取自 $W_\infty := [1, W] \cup \{0, \infty\}$。目标是找到顶点的一个划分 $V_1, \ldots, V_k$，使得对于所有 i，$|V_i| \leq \lceil |V|/k \rceil$ 成立，并使被该划分切割的边的权重最小化。更形式化地说，我们要最小化 $\text{cut}(V_1, \ldots, V_k) := \sum_{i=1}^{k} \sum_{\{u, v\} \in E \cap (V_i \times (V \setminus V_i))} \text{cap}(u, v)$。

我们指出，这个问题在理论上 [5,32–34] 以及实践中 [18,28,44,55], 都具有重要意义；平衡图划分算法常被用作大规模数据分析的预处理步骤。在应用领域中，取得该问题的实际改进颇受关注 [18] 例如，广受欢迎的 METIS 启发式算法 [44]已有 1,400+ 次引用。

由于上述问题即使在树上也难以在任意 $\epsilon > 0$ 下以 $n^{1-\epsilon}$ 的因子进行近似（ [34], ），我们考虑双准则近似算法。给定一个无向加权图 $G =$（V，E，容量），若将 V 划分为 V_1，.....，V_k，则当对所有 i 都有 $|V_i| \leq \beta\lceil n/k \rceil$，且 割（$V_1$，.....，$V_k$）$\leq \alpha \cdot$割(OPT) 时，该划分是一个双准则（$\alpha$，$\beta$）-近似解，其中 OPT =

（V_1^*，……，V_k^*）是在对所有 i 都满足 $|V_i^*| \le \lceil n/k \rceil$ 的条件下的最优解。我们注意到，前述困难性结果意味着：除非 P = NP.[1]，任何计算双准则（α，$1+\epsilon$）-近似的算法，只要其运行时间仅取决于 n 的多项式，就必须具有依赖于 $1/\epsilon$ 的超多项式运行时间，其中 $\alpha \ge 1$。

我们在以下定理中给出了静态情形的主要结果。该结果针对这一问题，给出了首个近似比为多对数级、且运行时间为近线性的算法。更具体地说，对于常数 k，我们可以在近线性时间内计算一个双准则 $(O(\log^4 n)\ 1+\epsilon)$-近似。作为比较，多项式时间算法 [34] 所能达到的最佳近似比为双准则 $(O(\log^{1.5} n \log\log n)\ 1+\epsilon)$-近似，运行时间为 $\Omega(n^4)$。

▶ **定理 2.** 设 $\epsilon > 0$ 和 $k \in \mathbb{N}$。令 $G = (V, E,$ 容量$)$ 为一个具有 n 个顶点和 m 条边、边权重取值于 W_∞的无向加权图。那么，对于 k-平衡划分问题，我们可以计算：

- 一个 $(O(\log^4 n)\ 1+\epsilon)$-近似，运行时间为 $(k/\epsilon)^{O(\log(1/\epsilon)/\epsilon)} \cdot O'(m \cdot \log^2(W)) + (k/\epsilon)^{O(1/\epsilon^2)}$。[2]
- 若 G 是一棵高度为 h的树，则可在 $(k/\epsilon)^{O(\log(1/\epsilon)/\epsilon)} \cdot O'(n \cdot h^2 \cdot \log^2(W)) + (k/\epsilon)^{O(1/\epsilon^2)}$ 时间内得到一个 $(1+\epsilon, 1+\epsilon)$-近似。若 G是一棵树，则可在 $(k/\epsilon)^{O(\log(1/\epsilon)/\epsilon)} \cdot O'(n^4 \cdot \log^2(W)) + (k/\epsilon)^{O(1/\epsilon^2)}$ 时间内得到一个 $(1, 1+\epsilon)$-近似。

此外，我们将结果扩展到动态设置，其中图 G 会经历边插入与删除。在下述定理中，我们给出该问题的首个更新时间为次多项式的动态算法。我们再次考虑双准则近似算法，其更新时间和查询时间依赖于 $1/\epsilon$，且这种依赖是超多项式的；这是无法避免的，因为如果我们能针对任意 $\alpha \ge 1$ 计算 $(\alpha, 1)$-近似解，或者对 $1/\epsilon$的依赖是多项式的，那么上文的困难性结果就意味着我们的更新时间和查询时间必须是 n 的超多项式（除非 P = NP）。

▶ **定理 3**。 设 $\epsilon > 0$ 和 $k \in \mathbb{N}$。设 $G = (V, E,$ 容量$)$ 为一个包含 n 个顶点、正在经历边插入与删除的无向加权图。那么，对于 k-平衡划分问题，我们可以维护：

- 若 G 是无权图，则可维护一个 $(n^{o(1)}, 1+\epsilon)$-近似解，其摊销更新时间为 $(k/\epsilon)^{O(\log(1/\epsilon)/\epsilon)} \cdot n^{o(1)} \cdot O'(\log^2(W))$，查询时间为 $(k/\epsilon)^{O(1/\epsilon^2)}$。若 G 是高度为 h的树，则可维护一个 $(1+\epsilon, 1+\epsilon)$-近似
- 解，其最坏情况更新时间为 $(k/\epsilon)^{O(\log(1/\epsilon)/\epsilon)} \cdot O'(h^3 \cdot \log^2(W))$，查询时间为 $(k/\epsilon)^{O(1/\epsilon^2)}$。

我们的方法受到 Feldmann 和 Foschini [34]所提出的动态规划的启发。然而， [34] 的算法中的 DP 行并不单调，因此，他们的动态规划无法直接通过我们的方法加速。因此，我们首先对 Feldmann 和 Foschini 的精确动态规划进行简化和推广，使其变为单调的。我们最终得到的动态规划仍然略微复杂，无法直接适用于我们的黑盒框架，但我们证明，仍然可以利用该框架中的思想来得到相应结果。在第 4.1 节中，我们将提供技术概述。

同样，我们的算法所维护的解可能在每次更新后都发生重大变化。与上述情况类似，我们在附录中证明（定理 35），在考虑次多项式更新时间时，这种情况无法避免。

---

[1] 如果我们有一种算法，能够在 poly$(n, 1/\epsilon)$ 时间内计算双准则 $(\alpha, 1+\epsilon)$-近似，那么我们可以设置 $\epsilon = 1/(2n)$，这意味着所有划分的大小都为 $\lceil n/k \rceil$。因此，我们可以在 poly(n) 时间内计算双准则 $(\alpha, 1)$-近似解，这与困难性结果相矛盾，除非 P = NP。[2] 我们使用记号 $O'(\cdot)$ 来省略 poly$(\log\ n, k, \log(1/\epsilon), \log\log(W))$ 中的因子。

同时源定位问题的结果。接下来，我们为 Andreev、Garrod、Golovin、Maggs 和 Meyerson 提出的同时源位置问题提供高效算法 [4]。该问题的输入是一个无向图 $G=(V, E)$，容量函数 cap d)，其边上有容量函数 cap：$E \to W_\infty$，顶点上有需求函数 d: $V \to W_\infty$。目标是选取一个最小的集合 $S \subseteq V$，其中的源点能够同时满足所有顶点需求。更具体地说，若存在一股从 S 中的顶点流出的流，能够向所有顶点 $v \in V$ 提供需求 $d(v)$，且不违反边上的容量约束，则源点集合 S 是可行的。目标是找到一个大小最小的可行源点集合。

我们将再次考虑双准则近似算法。设 S^* 是同时源位置问题的最优解。那么，如果 $|S| \leq \alpha |S^*|$，且当所有边容量都增加 β倍时， S是一个可行的源点集合，我们就称 S 为双准则(α， β)近似解。

以下定理概述了我们的主要结果。它给出了第一个用于同时源定位的近线性时间算法，该算法能计算一个 $(1+\epsilon)$-近似解，同时仅以 $O(\log^4 n)$ 倍数超出边容量。相比之下，具有任意多项式处理时间的最佳算法可在 $\Omega(n^3)$ [4]时间内计算一个双准则 $(1, O(\log^2 n \log\log n))$-近似解。

▶ 定理 *4.* 设 $\epsilon > 0$。设 $G=(V, E,$ 容量 $d)$ 是一个具有n 个顶点和 m 条边的无向加权图。那么，对于同时源位置问题，我们可以计算：一个 $(1+\epsilon, O(\log^4(n)))$-近似解，时间为[3] $\tilde{O}(\frac{1}{\epsilon^2} m)$。如果 G 是高度为 h的树，则可在时间 $\tilde{O}(\frac{1}{\epsilon^2} h^2 \cdot n)$内计算一个 $(1+\epsilon, 1)$-近似解。

接下来，我们转而讨论该问题的动态版本。我们考虑以下更新操作：设置需求(v, d)：将顶点 v 的需求更新为 $d(v)=d$，设置容量((u， v)， c)：将边 (u， v) 的容量更新为 cap(u， v) $=c$，删除(u， v)：删除边 (u， v)，插入((u， v)， c)：插入容量为 cap(u， v) 的边 (u， v) $=c$。

我们为该问题提出了首批更新时间为次多项式的动态算法，其边容量超出的幅度仅为一个较小的次多项式因子。

▶ **定理 5.** 设 $\epsilon > 0$。设 $G=(V, E,$ 容量 $d)$ 是一个含有 n 个顶点和 m 条边、并执行上述更新操作的图。那么，对于同时源位置问题，我们可以维护：

- 若所有边的容量均为 *1*，则可维护一个 $(1+\epsilon, n^{o(1)})$-近似解，摊销更新时间为 $n^{o(1)}/\epsilon^2$，预处理时间为$O(n^2/\epsilon^2)$。若仅允许更新操作 *SetDemand*(v, d)，则可维护一个 $(1+\epsilon, O(\log^4(n)))$-近似解，最坏情况更新时间为 $\tilde{O}(1/\epsilon^2)$，预处理时间为$\tilde{O}(m)$。若仅允许更新操作 *SetDemand*(v, d)，则可维护一个 $(1+\epsilon, O(\log^2(n) \log\log(n)))$-近似解，最坏情况更新时间为 $\tilde{O}(1/\epsilon^2)$，预处理时间为 *poly*(n)。若 G 是一棵高度为 h的树，则可维护一个 $(1+\epsilon, 1)$-近似解，最坏情况更新时间为 $\tilde{O}(h^3/\epsilon^2)$，预处理时间为 $O(n^2/\epsilon^2)$。

为获得这些结果，我们采用了与 Andreev 等人所用方法类似的动态规划方法 [4]。有趣的是，我们使用的动态规划函数实质上计算的是 Andreev 等人所用函数的反函数。我们在第 4.2 节概述这种方法的细节。进行这些修改后，这些定理便可直接应用我们用于维护具有单调行的动态规划的数据结构来证明。

[3] 我们用 $\tilde{O}(f(n, \epsilon, W))$ 表示形如 $f(n, \epsilon, W) \cdot \text{polylog}(n, \epsilon, \log W)$ 的运行时间。

论文结构。 第 2 节详细介绍我们针对具有单调行的动态规划所提出的条件。第 3 节展示我们关于 0-1 背包的结果，这些结果很好地说明了我们第 2 节黑盒框架的适用性。第 4 节对我们关于 k-平衡图划分和同时源点定位的更复杂结果作技术概述。我们在附录 A 概述附录的内容。此外，我们还在附录中介绍更多相关工作，并给出结果的完整证明。正文中省略的证明见附录 H。

开放问题与未来工作。 未来，我们可以利用这一框架，基于已有的动态规划设计更多动态算法，这将是一个值得探讨的方向，无论在理论上还是实践中都是如此。此外，一个引人入胜的问题是，定义 8 中的判据能否推广。事实上，我们的方法围绕着用分段常值函数逼近单调函数展开，而分段常值函数可以看作分段零次多项式。一个有趣的问题是，若用更高次多项式的分段（例如样条函数）来逼近动态规划行，我们能否得到更一般的判据？这一方向或许可以取得成果；例如，在附录 G 中，我们针对函数包含少量非单调性的情形给出了一个附带结果，并为 ℓ_∞-项链问题推导出一种动态算法。

2 单调动态规划表的维护

本节将介绍带有单调行 的动态规划表这一概念，以及我们采用的其他技术假设。随后，我们将介绍用于高效维护满足这些假设的动态规划表的数据结构。在该数据结构中，我们将使用分段常值函数存储动态规划的各行，下面先介绍这种函数。

分段常数函数的列表表示。设 $t \in \mathbb{R}, W \in [1,\infty)$，并定义 $W_\infty := \{0\} \cup [1, W] \cup \{+\infty\}$。如果存在实数 $0 = x_0 < x_1 < x_2 < \cdots < x_p = t$ 和数值 $y_1, \ldots, y_p \in W_\infty$，使得在每个区间 $[x_{i-1}, x_i)$ 上，函数 f 都是常数，且其值为 y_i，则称函数 $f\colon [0,t] \to W_\infty$ 是具有 p 个分段的分段常数函数。更正式地说，对于所有$i \in \{1, \ldots, p\}$，我们都有：对于所有实数 $x \in [x_{i-1}, x_i)$，$f(x) = y_i$，且 $f(x_p) = y_p$。注意，我们需要满足条件 $f(x_p) = y_p$，以确保 f 在整个定义域上都有定义。

对于分段常数函数 f的列表表示，我们使用双向链表存储数对 $(x_1, y_1) \ldots, (x_p, y_p)$。我们还将数对 (x_i, y_i) 存储在按 x_i值排序的二叉搜索树中，这使我们能够在时间$O(\log\ p)$ 内计算所有 $x \in [0,t]$对应的函数值 $f(x)$。下文中，我们假设所考虑的所有分段常数函数都以列表表示的形式存储，并额外配有一棵二叉搜索树。

我们使用的一项主要观察是：如果分段常数函数只有少量分段，那么对其进行的许多操作都能高效完成。下面的引理表明，若干操作的计算时间几乎与函数的分段数量呈线性关系，而不取决于其定义域的大小 f。[4] 对于 $\delta > 0$ 和 $y \in W_\infty$，我们用 $\lceil y \rceil_{1+\delta}$ 表示不小于 y 的最小的 $1+\delta$ 的幂，即 $\lceil y \rceil_{1+\delta} = \min\{(1+\delta)^i : (1+\delta)^i \geq y,\ \ i \in \mathbb{N}\}$；按照约定，$\lceil 0 \rceil_{1+\delta} = 0$，且 $\lceil \infty \rceil_{1+\delta} = \infty$。

▶ 引理 6。设 $t \in \mathbb{R}$ 和 $c \in \mathbb{R}_+$。设 $g, h\colon [0,t] \to W_\infty$ 为单调分段常数函数，分别有 p_g 和 p_h 个分段。那么，我们可以计算以下函数：$f_{\min}(x) := \min\{g(x)\ h(x)\}$，至多包含 $p_g + p_h$ 个分段，所需时间为 $O((p_g + p_h)\ \log(p_g + p_h))$；

[4] 我们注意到，这些操作本身可以在线性时间内完成。然而，由于我们还将列表表示中的数对 $(x_i,\ y_i)$ 存储在二叉搜索树中，引理中的运行时间还包含一个额外的对数因子。

- $f_{\text{shift}}(x) := g(x-c)$，当 $x \geq c$，$f_{\text{shift}}(x) = g(0)$ 时，至多包含 p_g 个分段，所需时间为 $O(p_g \log(p_g))$；$f_{\text{add}}(x) := g(x) + h(x)$，至多包含 $p_g + p_h$ 个分段，所需时间为 $O((p_g + p_h)\log(p_g + p_h))$；$f_{\text{round}}(x) := d g(x) e_{1+\delta}$，当 $\delta > 0$ 时，至多包含 $2 + d\log_{1+\delta}(W) e$ 个分段，所需时间为 $O(p_g \log(p_g))$。

注意，如果我们设定 $\tilde{f} = \lceil f \rceil_{1+\delta}$，那么 $\tilde{f}$ 按以下意义是 $1+\delta$ 的 $(1+\delta)$-近似：对于 $\alpha > 1$，如果对于所有 $x \in [0,t]$，函数 $f\colon [0,t] \to W_\infty$ α-近似函数 $\tilde{f}\colon [0,t] \to W_\infty$，我们就称函数 $\tilde{f}$ 是函数 f 的 $(1+\delta)$-近似。

$$f(x) \leq \tilde{f}(x) \leq \alpha \cdot f(x). \tag{1}$$

此外，如果 f 是单调的，那么舍入后的函数 $\tilde{f}$ 至多包含 $O(\log_{1+\delta}(W))$ 个分段。这一点在后文中至关重要，因为它保证了如果我们对动态规划表的每一行各执行一次舍入操作，所得函数将只有很少的分段，并且可以高效地对该函数执行操作。

接下来，考虑函数 f_1，$f_2\colon [0,t] \to W_\infty$。如果对于所有 $x \in [0,t]$，$f(x) = (f_1 \oplus f_2)(x) := \min_{\bar{x} \in [0,x]} f_1(\bar{x}) + f_2(x - \bar{x})$，则函数 $f\colon [0,t] \to W_\infty$ 是 $f_1 \oplus f_2$ 的 $(\min,+)$-卷积。这种卷积对于许多动态规划的计算非常有用。下面的引理表明，我们可以高效地计算分段常数函数的卷积。

▶ 引理 7. 设 $f_1, f_2\colon [0,t] \to W_\infty$ 为至多包含 p 个分段的分段常数函数，并假设其中一个函数单调递减。那么，我们可以在 $O(p^2 \log p)$ 时间内计算函数 $f\colon [0,t] \to W_\infty$，其中 $f = f_1 \oplus f_2$，且 f 是一个包含 $O(p^2)$ 个分段的分段常数函数。此外，计算出 f 后，对于任意 $x \in [0,t]$，我们都可以在 $O(\log p)$ 时间内返回一个值 $\bar{x}^* \in [0,t]$，使得 $f(x) = f_1(\bar{x}^*) + f_2(x - \bar{x}^*)$。

现在请注意，引理 7 对我们的方法有一个缺点：每应用一次，分段的数量（即函数的复杂性）都会呈二次增长。一个有助于缓解这一问题的重要性质是，卷积的结果仍然是单调函数，这一点我们将在附录中的引理 22 证明。之后，为了使函数中的分段数量保持较少，每次通过引理 7 执行卷积（这可能导致分段数量呈二次增长）后，我们都会进行舍入操作 $\lceil \cdot \rceil_{1+\delta}$（参见引理 6）。这会使近似损失一个因子 $1+\delta$，但能保证所得函数包含 $O(\log_{1+\delta}(W))$ 个分段。这对于确保我们的函数仅有少量分段至关重要。

维护具有单调行的动态规划。接下来，我们正式介绍动态规划方案。我们考虑具有有限行集 $\mathcal{I}$ 和列集 $\mathcal{J}$ 的动态规划表，其条目取值于 W_∞。我们将动态规划表视为函数 $\text{DP}: \mathcal{I} \times \mathcal{J} \to W_\infty$.[5] 此外，我们将动态规划表的第 i 行与函数 $\text{DP}(i,\cdot)\colon \mathcal{J} \to W_\infty$ 相关联，并使用前文介绍的分段常数函数来存储每个这样的函数 $\text{DP}(i,\cdot)$。

接下来，我们介绍 DP 行的依赖图。更具体地说，依赖图 $D = (\mathcal{I}, E_D)$ 是一个有向图，其顶点为行 $\mathcal{I}$；如果对于某些列 $j, j' \in \mathcal{J}$，计算 $\text{DP}(i',j')$ 需要用到 $\text{DP}(i,j)$，那么这两行之间就存在一条有向边 (i',i)。我们用 $\text{In}(i) = \{i' \in \mathcal{I}\colon (i',i) \in E_D\}$ 表示计算行 i' 所需的行集合。本文余下部分假设依赖图是一个有向无环图，这适用于我们研究的所有应用。我们还用 $\text{Reach}(i)$ 表示从图 D 中的行 i 可达的顶点集合。

[5] 尽管我们的定义可能让人以为我们只考虑二维动态规划表，但我们并不要求 $\mathcal{I}$ 具有序关系，并且允许 $\mathcal{I}$ 为任意有限集。例如，在 D 节中，我们将令 $\mathcal{I}$ 为与四维动态规划参数对应的三元组。

10 单调动态规划的动态维护及其应用

由于我们假设依赖图是一个有向无环图，因此一旦计算出 In(i) 中各 DP 行的解，就可以计算第 i 行 DP。我们假设这是通过一个过程 P_i 完成的：该过程以所有 $i' \in$ In(i) 对应的 DP 行 DP(i',·) 作为输入，并返回 DP 行 DP(i,·) $= \mathcal{P}_i(\{\text{DP}(i', \cdot)\colon \ i' \in \text{In}(i)\}$</style)。

接下来，我们介绍用于说明本方案何时适用的条件。在定义中以及本文余下部分，我们用 ADP 表示近似动态规划表，它近似于精确动态规划表 DP。令 $\beta > 1$。若对于所有 $i \in \mathcal{I}$ $j \in \mathcal{J}$，均有 DP(i, j) $\leq$ ADP(i, j) $\leq \beta$DP(i, j)，则称 ADP β-近似于 DP。

▶ 定义 *8.* 若动态规划表满足以下条件，则称其为 *(h, α, p)*-良好态：*1.*（单调性：）对于所有 $i \in I$，函数 *DP(i, ·)* 都是单调的。*2.*（依赖图：）依赖图是一个有向无环图，且对于所有 $i \in I$，$|$ *Reach(i)*$| \leq h$。*3.*（敏感性：）假设 $\beta > 1$，并且对于所有 $i' \in$ *In(i)*，我们得到 *DP(i', ·)* 的一个 β-近似 *ADP(i', ·)*。那么，在 *ADP(i', ·)* 上应用 P_i，可得到 *DP(i, ·)* 的一个 β-近似，即

$$\text{DP}(i, \cdot) \leq \mathcal{P}_i(\{\text{ADP}(i', \cdot)\colon i' \in \text{In}(i)\}) \leq \beta \cdot \text{DP}(i, \cdot).$$

˜*4.*（分段：）对于每个过程 P_i，都存在一个近似过程 P_i，使得：˜*(a)* P_i*({ADP(i', ·) :* $i' \in$ *In(i)}) 是 P_i({ADP(i', ·) :* $i' \in$ *In(i)}) 的 α-近似；˜*(b)* P_i 可以表示为由引理 *6* 中的常数个操作以及至多一次引理 *7* 的应用所构成的复合；并且˜*(c)* P_i 返回一个至多有 p 个分段的单调分段常数函数。

该定义的动机如下：当分段常数函数是单调的且分段数较少时，我们对这些函数执行的操作具有高效的运行时间。性质 (1)、4(b) 和 4(c) 保证了这一点。其次，如果每行最多只能到达 h 个其他行，且满足敏感性条件，那么舍入误差就不会累积过多。性质 (2)、(3) 和 4(a) 保证了这一点。

尽管该定义乍看之下可能略显技术性，但它适用于许多场景。特别地，当依赖图是一棵高度为 h 且所有边都指向根节点的有根树时，性质 (2) 得以满足；我们的所有应用都属于这种情况。我们的第 3 节 0-1 背包动态规划以及第 E 节同时源定位动态规划都立即满足其他条件。然而，我们的平衡图划分动态规划违反了定义 8 的性质 (4b)。因此，我们还将在 C.2 节中考虑一种较弱的假设，但该假设将无法得到如下定理 9 和定理 10 所示的良好黑盒结果。

接下来，我们陈述主要结果。这些结果表明，对于 (O(1),ln(1+ϵ)/O(1) O(1))-良好动态规划，我们可以获得在近线性时间和空间内运行的静态 $(1+\epsilon)$-近似˜ ˜ ˜算法。它们还表明，在这一假设下，我们可以动态维护 $(1+\epsilon)$-近似动态规划解，且更新时间为多对数级。

静态算法的主要定理如下。

▶ **定理 9.** 考虑一个(h, α, p)-良好态动态规划。于是，我们可以在时间和空间$(|I| \cdot p^2 \log(p))$内计算出一个近似动态规划表ADP，它以 α^{h+1}近似动态规划。

稍后，我们将把该定理应用于依赖树高度为对数级的动态规划$h = O(\log n)$，将近似比设为 $\alpha = \ln(1+\epsilon)/(h+1)$，并将分段数设为 $p = \text{polylog}(W)$。由此将得到我们所需的算法，其运行时间˜接近线性，为 $O(|\mathcal{I}|)$，且空间使用量也较低。请注意，这大幅改进了蛮力算法的运行时间和空间开销 $\Omega(|\mathcal{I}| \cdot |\mathcal{J}|)$。

该定理的证明基于以下观察：在依赖图中从一个顶点移动到另一个顶点时，近似比会损失一个乘法因子 α；由于每个顶点最多只能到达 h 个其他顶点，这种损失累积后至多为 α^{h+1}。˜结合对函数$\mathcal{P}_i$ 的假设以及引理 6 和引理 7 的结果，我们可在 $O(p^2 \log(p))$ 时间内计算每一行 ADP$(i, \cdot)$，总运行时间为 $O(|I| \cdot p^2 \log(p))$。

我们还给出了以下针对动态设置的扩展，它表明，如果某个 DP 行发生变化，我们就能高效地更新整个表。

▶ **定理 10**。考虑一个(h, α, p)-良好态动态规划，并假设行 i 发生了变化。然后，我们可以更新近似动态规划表 ADP，使其在耗时 $O(h \cdot p^2 \log(p))$ 后成为DP的 α^{h+1}-近似。

与之前一样，我们通常会在 $h = O(\log n)$、$\alpha = \ln(1+\epsilon)/(h+1)$ 和$p = \text{polylog}(W)$ 的取值下使用该定理。这样就能得到我们所需的多对数更新时间。请注意，与使用二维数组存储动态规划表相比，这显著提升了速度：在那种情况下，即使更新单独一行 也需要 Ω(|$\mathcal{J}$|) 的时间，而在许多应用中，这已经相当于输入规模的线性时间。

根据引理 6 和引理 7，只需注意到，在某一行 i 发生变化后，我们只需更新依赖图中可从 i 到达的那些行。但这些行至多有 h行，且每一行都可以在 $O(p^2\log(p))$ 时间内更新。

3 全动态背包

在 *0-1* 背包问题中，输入包括一个背包容量 $B \in \mathbb{R}_+$ 和一组 n 个物品，其中每个物品 $i \in [n]$ 都有一个权重 $w_i \in \mathbb{R}_+$ 和一个价格 $p_i \in [1, \infty)$。目标是在满足约束 $\sum_{i \in I} w_i \leq B$的同时，找到一组物品 I，使 $\sum_{i \in I} p_i$ 最大化。对于一组物品 $I \subseteq [n]$，我们将总和 $\sum_{i \in I} w_i$ 称为 的权重。

在本节余下部分，我们令 $W = \sum_i p_i$ 和 $t = \sum_{i \in [n]} w_i$。

接下来，我们首先推导出一种动态算法，其更新时间为 $\tilde{O}(\log^3(n) \log^2(W)/\epsilon^2)$，该算法基于我们针对具有单调行的动态规划所构建的框架。然后，我们将在第 3.2 节中将该算法用作子程序，以获得更新时间为 $\tilde{O}(\log^2(nW)/\epsilon^2)$ 的更快算法；这将证明定理 1。

▶**定理 1.** 设 $\epsilon > 0$。存在一种全动态背包问题算法，能够维护一个$(1+\epsilon)$-近似解，其最坏情况更新时间为 $\frac{1}{\epsilon^2}\log^2(nW)\,\text{polylog}\left(\frac{1}{\epsilon}\log(nW)\right)$。

下面我们还将证明，可以在时间 $O(|I|)$ 内返回维护的解 I，并且可以在时间 $O(1)$ 内回答给定物品 $i \in [n]$ 是否包含在 I 中的查询。这与 [29] 的查询时间相匹配。

3.1 通过单调函数卷积求解背包问题

首先，我们简要回顾 Chan [21] 提出的背包问题方法。我们考虑近似函数 f_J 的更一般问题：$[0, t] \to \mathbb{R}_+$，其中 $J \subseteq [n]$ 是物品集合，并且

$$f_J(x) = \max\left\{\sum_{i \in I} p_i \colon \sum_{i \in I} w_i \leq x,\ I \subseteq J\right\}. \tag{2}$$

12 单调动态规划的动态维护及其应用

直观地说，值 $f_J(x)$ 对应于这样一个最佳背包解：我们只能选择包含在 J 中的物品，并且该解的权重至多为 x。因此，$f_{[n]}(B)$ 对应于全局背包实例的最优解。

注意，对于每个 $J \subseteq [n]$，$f_J(x)$ 都是单调递增的分段常数函数：事实上，考虑 $x' \leq x$。任何对 x' 可行的解 $I \subseteq J$（即 I 的权重至多为 x'）对于 x 也都是可行解。因此，$f_J(x') \leq f_J(x)$，所以 f_J 是单调递增的。此外，f_J 是分段常数函数，因为每个函数值 $f_J(x)$ 都对应于一个解 $I \subseteq J$，而 $I \subseteq J$ 的选择数量是有限的。

接下来注意，如果我们有两个不相交子集 J_1，$J_2 \subseteq [n]$，那么 $f_{J_1 \cup J_2}$ 是 f_{J_1} 和 f_{J_2} 的（max $+$）-卷积，即对于所有 x，都有

$$f_{J_1 \cup J_2}(x) = \max_{\bar{x}} f_{J_1}(\bar{x}) + f_{J_2}(x - \bar{x}).$$

观察可知，对于每个 x，权重至多为 x 的实例 $J_1 \cup J_2$ 的最优解 I 可以拆分为两个互不相交的解 $I_1 \subseteq J_1$ 和 $I_2 \subseteq J_2$，使得 I_1 的权重为 $\bar{x}$，而 I_2 的背包权重至多为 $x - \bar{x}$（适当选择 $\bar{x} \in [0, x]$ 即可）。由此可得，若有两个分别包含互不相同物品集合 J_1 和 J_2 的背包实例，那么只需计算 f_{J_1} 和 f_{J_2} 的（max $+$）卷积，即可求出物品集合为 $J_1 \cup J_2$ 的背包实例的解。

精确动态规划。前面的段落说明了一种计算背包实例精确解的简单方法：对每个物品 $i \in [n]$，计算函数 $f_{\{i\}}$，然后递归合并大小为 2^j、$j = 1$、.....、$\lceil \log n \rceil$ 的集合对应的解，通过计算（max $+$）卷积，直至求得全局解 $f_{[n]}$。我们使用平衡二叉树递归合并这些解，得到一棵高度为 O（log n）的树。

更具体地说，我们构建一棵有根平衡二叉树 T，其中有 n 个叶节点，所有边都指向根节点。每个物品 i对应一个叶节点 $f_{\{i\}}$。T 中的每个内部节点 u都按照公式 (2) 与一个函数 f_{J_u} 相关联，其中 J_u 是以 u为根的子树中所有物品的集合。为简化记号，我们也将f_{J_u} 称为 f_u。

现在我们考虑动态规划的精确计算。这将揭示定义 8 中的过程$\mathcal{P}_i$。作为基本情形，对于每个 $i \in [n]$，T 的第 i个叶节点包含函数 $f_{\{i\}}$；这是一个分段常数函数，在区间 $[0, w_i)$上的取值为 0，在区间 $[w_i, t]$上的取值为 p_i。

接下来，对于 T 中的每个内部节点 u，若其子节点为 u_1 和 u_2，我们将 f_u 设为 f_{u_1} 与 f_{u_2}的 (max $+$)-卷积。根据归纳法可知，对于 T中的每个节点 u，都有 $J_u = J_{u_1} \cup J_{u_2}$，因此 J_u 是所有对应叶节点位于子树 T_u中的物品的集合。因此，对于 T 的根节点 r，有 $f_r = f_{[n]}$，而 $f_r(B)$是全局背包实例的最优解。

接下来，我们验证我们的动态规划满足定义 8 的性质 (1–3)。

首先，注意到上文中的树 T 也是我们动态规划的依赖图。因此，我们的动态规划为 T 的每个顶点都设置了一行，共有 $O(n)$ 行。此外，由于 T的高度为 $O(\log n)$，且所有边都指向根节点，因此每个顶点至多可以到达$h = O(\log n)$ 个顶点。因此，满足定义 8 的性质 (2)。

其次，我们注意到，在上述两种情况下，与动态规划表各行对应的函数 $f_{\{i\}}$ 和 f_u 都是单调递增的（我们上文已经论证了所有函数f_J均如此）。因此，满足性质 (1)。

第三，注意到性质 (3) 也满足，因为 (max $+$)-卷积满足我们的敏感性条件。

我们得出结论，定义 8 的前三个性质均已满足。遗憾的是，这尚不能说明我们可以获得高效算法：注意，如果我们自底向上计算精确动态规划，那么每个节点都要计算一次卷积，因此这种方法的总运行时间为 $O(n \cdot t(p))$，其中 p 是函数中分段数量的上界，而 $t(p)$ 是计算两个各含 p 个分段的函数的 (max $+$)-卷积所需的时间。不过，值得注意的是，计算这些卷积可能会耗费大量时间，因为每次卷积后，函数的分段数量可能会呈二次增长（见引理 7）。我们将在下文通过舍入来解决这一问题。

近似动态规划。 接下来，我们考虑近似，这将揭示定义 8 中的函数$\tilde{\mathcal{P}}_i$。

首先，注意我们需要高效地计算单调递增函数的 (max $+$)-卷积。我们观察到，可以利用引理 7 中用于计算单调递减函数的 (min $+$)-卷积的子程序来高效完成这一计算：事实上，假设 f 是两个单调递增函数 g 和 h的 (max $+$)-卷积，那么对所有 x，都有

$$f(x) = \max_{\bar{x}}\{g(\bar{x}) + h(x-\bar{x})\} = -\min_{\bar{x}}\{-g(\bar{x}) + (-h(x-\bar{x}))\}.$$

现在注意到，$-g$ 和 $-h$ 是单调递减函数，因此，$f = -((-g) \oplus (-h))$，其中 $\oplus$ 表示 (min $+$)-卷积。因此，我们可以使用引理 7 中计算 (min $+$)-卷积的高效算法，其运行时间相同。[6]

现在我们可以定义子程序$\tilde{\mathcal{P}}_i$。令 $\delta > 0$ 为稍后设定的参数。每当我们通过（max $+$）-卷积计算函数 f_u 时，就使用引理 7 中的高效子程序。计算完卷积后，我们通过引理 6 中的子程序设置 $f_u = \lceil f_u \rceil_{1+\delta}$。

注意，这种方法满足定义 8 的性质 (4a)，其中 $\alpha = 1 + \delta$。此外，由于我们只使用一次卷积和一次舍入步骤，因此满足性质 (4b)。最后，结果函数是单调的，并且在舍入后具有 $p = O(\log_{1+\delta}(W))$，所以也满足性质 (4c)。

上述论证表明，对于 $h = \lceil \log\ n \rceil$、$\alpha = 1+\delta, \delta = \ln(1+\epsilon)/\lceil \log\ n \rceil$ 以及 $p = O(\log_{1+\delta}(W)) = O(\log(W)/\delta)$，我们的动态规划是 (h, α, p)-良好态的。因此，定理 10 立即推出以下引理。

▶ **引理 11.** 设 $\epsilon > 0$。存在一种算法，可在 $n \cdot \frac{1}{\epsilon^2}\log^2(n)\log^2(W) \cdot \mathrm{polylog}(\frac{1}{\epsilon}\log(nW))$ 的时间内，为 *0-1* 背包计算一个 $(1+\epsilon)$-近似解。

我们注意到，可以按如下方式在 $|I|\log(n) \cdot \mathrm{polylog}(\frac{1}{\epsilon}\log(nW))$ 时间内返回解 I。回想一下，我们的全局目标函数值由 $f_r(B)$ 达到，并且 $f_r(B) = f_{u_1}(\bar{x}^*) + f_{u_2}(B - \bar{x}^*)$，其中 u_1 和 u_2 是依赖树中根节点 r 下方的节点。现在利用引理 7 的第二部分，我们可以在 $O(\log p)$ 时间内得到 $\bar{x}^*$ 的值。如果 $f_{u_1}(\bar{x}^*) > 0$，我们就递归处理 $f_{u_1}(\bar{x}^*)$；如果 $f_{u_2}(B - \bar{x}^*) > 0$，我们也递归处理 $f_{u_2}(B - \bar{x}^*)$。最终我们会到达叶节点 i，并且当且仅当 $f_{\{i\}}(x) > 0$ 时，将 i 纳入解中。注意，由于我们只对严格大于零的函数值进行递归，因此对于每个纳入解中的物品，我们都只需沿依赖树中的一条路径前进，该路径的高度为 $O(\log n)$；而在每个内部节点上的工作量至多为

[6] 我们注意到，严格来说，引理 7 只能应用于取非负值的函数。不过，可以通过给 g 和 h 加上一个数 C 来实现，其中 C 是 g 和 h 所取值的一个上界；最后再减去常数函数 $2C$，即令 $f = -((-g+C) \oplus (-h+C)) - 2C$。

$O(\log\ p)$。因此，总时间为 $O(|I|\log(n)\log(p))$，结合我们对上文中 p 的选择，即可得出结论。

扩展到动态设置。 接下来，我们将结果扩展到动态设置。为简单起见，我们假设 n 是可用物品（ S中的物品）最大数量的上界，并在开始时将其提供给我们的算法。[7] 我们考虑从集合中插入和删除物品的更新操作。具体来说，我们考虑以下更新操作：

- 插入(p_i, w_i)，其中将 i 添加到 S ，方法是将物品 i 的价格和权重分别设为 $p_i \in W_\infty$ 和 $w_i \in \mathbb{R}_+$；以及删除(i)，即从物品集合中移除物品 i 。

我们的实现如下。在预处理阶段，我们构建与上文相同的树 T，并使用上文的子程序计算函数 $f_{\{i\}}$。对于操作删除(i)，我们设置 $p_i = 0$ 和 $w_i = 0$，这恰好会改变动态规划表中的一行。对于操作插入(p_i, w_i)，我们分别将物品 i 的价格和权重设为 p_i 和 w_i，这同样会改变动态规划表中的一行。修改该行后，我们通过定理 10 重新计算全局动态规划解。由于动态规划具有与上文相同参数的(h, α, p)良好态，定理得出以下命题。

▶命题 12. </style}设 。对于完全动态背包问题，存在一种算法，能够维护一个$(1+\epsilon)$近似解，其最坏情况更新时间为$\frac{1}{\epsilon^2}\log^3(n)\log^2(W)\cdot\operatorname{polylog}\left(\frac{1}{\epsilon}\log(nW)\right)$。

注意，使用与静态算法相同的过程，我们可以在 $|I|\log(n)\cdot\operatorname{polylog}(\frac{1}{\epsilon}\log(nW))$ 时间内返回解 I。此外，给定一个物品 $i \in [n]$，我们可以在 $\log(n)\cdot\operatorname{polylog}(\frac{1}{\epsilon}\log(nW))$ 时间内返回 $i \in I$ 是否成立。这可以通过使用与静态情形相同的查询过程来完成：我们只在依赖树中包含物品 i 对应节点的唯一子树上递归。

我们注意到，上述命题在对 $\frac{1}{\epsilon}$ 的依赖方面，已经优于 Eberle 等人结果中的更新时间 [29] ；但它对 $\log(nW)$ 的依赖更差。不过，我们的查询时间比 [29] 中耗时 $O(1)$ 的查询操作更慢。我们将在下一小节解决这些问题，届时会将命题 12 中的算法用作子程序。

3.2 动态维护一个小实例

接下来，通过结合命题 12 中的算法与 Eberle 等人 [29] 的思想，我们获得一种更新时间为 $\tilde{O}(\frac{1}{\epsilon^2}\log^2(nW))$ 的更快动态算法。我们的总体方法如下。首先，我们将物品划分为少量的价格类别。然后，从每个价格类别中选取一些权重较小的物品。这将得到一个非常小的背包实例 X ，我们使用命题 12 中的子程序维护其近似最优解；由于该实例非常小（即 $|X| \ll n$），维护该实例的更新时间基本上变为 $O(\frac{1}{\epsilon^2}\log^2(W))$，也就是说，我们避免了命题中使更新时间过高的 $O(\log^3 n)$ 项。对于其余未包含在 X中的物品，我们证明可以使用分数背包计算出一个良好的解，

[7] 通过摊还分析论证，可以去掉这一假设。具体而言，每当物品数量少于 $n/2$ 或多于 n时，我们就用新的 n 重建数据结构。每次重建都可以在 $O(nt(n))$ 时间内完成，其中 $t(n)$ 是我们的更新时间。由于只有在发生 $\Omega(n)$ 次更新后才会进行重建，因此我们可以将这项代价摊还到上次重建以来发生的更新上。

而分数背包可以通过一组二叉搜索树轻松求解。接下来只需证明这两个解的组合是一个 $(1+\epsilon)$-近似解。

我们的算法与 Eberle 等人的算法主要有以下区别。Eberle 等人也将物品划分为少量价格类别。他们也会组合针对少量重物品的解，以及基于其余物品的分数背包解的解。不过，他们必须枚举许多不同的集合 X，还必须猜测分数背包解的近似价格；具体而言，他们会枚举$\Theta(\frac{1}{\epsilon^2}\log(W))$ 种选择来确定 X，而为分数背包解所需作出的猜测数量为 $\Theta(\frac{1}{\epsilon}\log(W))$。因此，他们必须考虑 $\Theta(\frac{1}{\epsilon^3}\log^2(W))$ 种猜测，并且对每种猜测都必须计算近似解。由于他们每次都必须从头运行静态算法，因此每种猜测所需时间为 $\Theta(\frac{1}{\epsilon^4})$。相比之下，我们只需考虑单个集合 X，并在命题 12 的数据结构中维护它，这为我们节省了大量时间。此外，我们用于存储 X解的分段常数函数，实质上“引导”了我们对分数背包解权重所作的 $\Theta(\frac{1}{\epsilon}\log(W))$ 种猜测。在分析中，我们必须更加谨慎一些，以确保对分数背包解权重的猜测能够保证正确的近似比。

定义。我们假设 $\epsilon < 1$，且 $1/\epsilon$ 是一个整数。更具体地说，我们运行算法，其中 $\epsilon' = \max\{\frac{1}{i} : \frac{1}{i} \le \epsilon,\ i \in \mathbb{N}\}$。设 $L = \lceil \log_{1+\epsilon}(W) \rceil$，并回顾一下我们设定了$W = \sum_i p_i$。

我们将价格类别 $V_\ell = \{i : (1+\epsilon)^\ell \le p_i < (1+\epsilon)^{\ell+1}\}$定义如下。下文中，我们假设价格类别 V_ℓ 中的所有物品价格都恰好为 $(1+\epsilon)^{\ell+1}$。作出此假设只会带来 $1+\epsilon$ 倍的损失。此外，我们令 $V_\ell^{1/\epsilon}$ 为从V_ℓ 中选出的 $1/\epsilon$ 个权重最小的物品的集合 w_i（任意打破平局）。我们还定义 $V_\ell' = V_\ell \setminus V_\ell^{1/\epsilon}$。

接下来，对于所有 $\ell \ge 0$，我们设置 $X = \bigcup_{\ell \ge 0} V_\ell^{1/\epsilon}$ 和 $Y = \bigcup_{\ell \ge 0} V_\ell'$。注意，$X$ 和 Y 将物品集合划分为两部分，并且$|X| = \frac{1}{\epsilon} \cdot L = O(\epsilon^{-2}\log(W))$。

现在，我们的策略是使用命题 12 中的算法，维护 X 中物品的一个解。然后，我们展示如何将 X 的解与基于分数背包和收费论证的 Y 解结合起来。

数据结构。对于每个 $\ell \in [L]$，我们按照权重的非递减顺序维护 V_ℓ。我们还在二叉搜索树中维护集合 X，按物品的索引对其排序，并对 X 中的物品维护命题 12 中的数据结构。

此外，令 $U_\ell = \bigcup_{\ell' \le \ell} V_{\ell'}'$ 表示所有不包含在 X 中且价格类别不超过 ℓ 的物品组成的集合。对于每个 ℓ，我们在二叉搜索树 T 中维护集合 U_ℓ，其中物品作为叶节点存储，并按其密度 $\frac{p_i}{w_i}$ 排序。在 T 的每个内部节点 u 中，我们存储以 u 为根的子树 T_u 中物品的总权重，以及 T_u 中物品的总利润。注意，这使我们能够在 $O(\log n)$ 时间内回答以下类型的查询：“给定预算 b，在 U_ℓ 中，权重不超过 b 的最优分数背包解的值是多少？”

更新。现在考虑一次物品插入或删除，并假设更新的物品属于价格类别 V_ℓ。我们首先更新集合 V_ℓ、$U_{\ell'}$（针对 $\ell' \le \ell$），以及集合 X 和 Y。注意，对于这些集合中的每一个，至多有一个物品会被删除或插入。因此，这些步骤可以在 $O(\ell \cdot \log(n)) = O(\epsilon^{-1}\log(W)\log(n))$ 的时间内完成。

接下来，如果 X 在上一步中发生了变化，我们也会执行相应的更新

[8] 在分数背包问题中，我们可以按比例使用物品。将物品按密度排序，然后贪心地将物品加入解中，直到用完预算 b，即可得到一个最优解。这种方法至多会按比例使用一件物品（即用完预算时所使用的那件物品）。

在命题 12 的数据结构中。由于根据 X 的构造， $|X| = O(\epsilon^{-2}\log(W))$ 成立，因此用于维护 X 的背包解的数据结构，其更新操作总共需要的时间为

$$O\left(\epsilon^{-2}\log^3(|X|)\log^2(W)\cdot \operatorname{polylog}\left(\frac{1}{\epsilon}\log(|X|W)\right)\right)$$
$$= O\left(\epsilon^{-2}\log^2(W)\cdot \operatorname{polylog}\left(\frac{1}{\epsilon}\log(nW)\right)\right).$$

此外，我们可以在$\epsilon^{-2}\log(W)\cdot\operatorname{polylog}(\frac{1}{\epsilon}\log(nW))$ 时间内明确写出针对 X 中物品的解 I_X ，因为 $|X| = O(\epsilon^{-2}\log(W))$。另外，对于每个 $i \in I_X$，我们可以设置一个位来指示 $i \in I_X$。注意，写出 I_X 并设置这些位所需的时间已包含在上述更新时间中。

查询。 返回解的值： 我们按如下方式返回全局背包解的值。

考虑命题 12 中的数据结构，它维护针对 X 中物品的一个解。注意，该解以具有 $p \le L$ 个分段的分段常数函数形式存储，并考虑该函数的列表表示 $(x_1, y_1)\dots,(x_p, y_p)$。

我们的策略如下：对于每个 $i = 1,\dots,p$，我们考虑一个解，该解在 X 中的物品上花费预算 x_i，并在 Y 中的物品上花费预算 $B - x_i$。然后，我们取所考虑的所有解中的最大值。更具体地说，给定 $i = 1,\dots,p$，我们按如下方式得到解。我们选取 ℓ_i，使得 $(1+\epsilon)^{\ell_i} = \lceil \epsilon\cdot y_i\rceil^{1+\epsilon}$（关于这一选择的理由，见下文引理 13）。现在，我们使用 U_{ℓ_i} 的二叉搜索树，查找在预算至多为 $b = B - x_i$ 的情况下，对 $U_{\ell_i} \subseteq Y$ 中的物品进行分数背包所能获得的最高利润。令 y'_i 为此次查询的值，扣除从（至多一个）分数切分物品中获得的利润后得到。我们还存储分数背包解中最后一个物品的密度。然后，我们返回所有 $i = 1,\dots,p$ 中 $y_i + y'_i$ 的最大值。

注意，由于 X 的解至多包含 $L = O(\epsilon^{-1}\log(W))$ 个分段，并且我们对每个分段只需在一棵二叉搜索树中执行一次查询，因此返回解的值总共需要 $O(\epsilon^{-1}\log(W)\log(n))$ 时间。注意，这一时间开销被更新时间所涵盖。

返回完整解：现在，我们可以按如下方式在 $O(|I|)$ 时间内返回全局解 I。注意， I 由 X 中物品的解 I_X 以及分数背包解中的物品组成。在更新过程中，我们已经存储了 I_X 中的物品，因此可以在 $O(|I_X|)$ 时间内将它们列出。接下来，要返回分数背包解中的物品，请回想一下，我们存储了分数背包解中最后一个物品的密度。因此，我们只需按密度非递减的顺序输出物品，同时确保其密度高于所需的密度阈值。完成这一操作所需的时间与分数背包解的大小呈线性关系。这本质上与 [29]中的查询过程相同。

返回物品是否属于解：此外，注意到上述结论意味着，我们可以在 $O(1)$时间内回答物品 $i \in [n]$ 是否包含在解中：如果$i \in X$，那么我们已经存储了一个比特，用于记录 $i \in I_X$ 是否成立。如果 $i \notin X$，那么我们可以通过检查其密度高于还是低于分数背包解中最后一个物品给出的阈值，判断 i 是否属于分数背包解。

分析。我们先对 OPT 作一些简化。令 OPT′ 表示 OPT 的一个版本，其中对于每个 $\ell \in [L]$，我们从V_ℓ中选取权重最小的 $|$个 OPT$\cap V_\ell|$ 物品。这只会损失一个 $1+\epsilon$因子。接下来，定义 OPT′$_X$ =OPT′$\cap X$和 OPT′$_Y$ =OPT′$\cap Y$。注意，根据我们选取 OPT′, 的方式，OPT′$_Y \cap V_\ell \neq \emptyset$ 当且仅当$|$OPT′$\cap V_\ell| > 1/\epsilon$。

令 p_X 表示 OPT′$_X$ 中物品的总价格，令 w_X 表示 OPT′$_X$ 中物品的总权重。令 f 表示存储物品解的分段常数函数

在 X中的物品。注意，根据命题 12，我们有

$$p_X \leq f(w_X) \leq (1+\epsilon)p_X.$$

此外，函数值 $f(w_X)$ 是片段 (x_{i^*}, y_{i^*}) 的一部分，其中有 $x_{i^*} \leq w_X$ 以及 $y_{i^*} = f(w_X)$。

下一个引理说明了我们为何在算法中将 ℓ_i 设置为使 $(1+\epsilon)^{\ell_i} = \lceil \epsilon \cdot y_i \rceil_{1+\epsilon}$。为此，令 ℓ_{i^*} 满足 $(1+\epsilon)^{\ell_{i^*}} = \lceil \epsilon \cdot y_{i^*} \rceil_{1+\epsilon}$，并令 ℓ_Y 为 OPT 中最有价值物品的价格类别。在该引理中，我们证明 $\ell_{i^*} \geq \ell_Y$。我们将利用这一点来证明，我们的 X 解的利润 y_{i^*} 足够高，因此可以将一个来自分数背包问题的分数切分物品计入 X 的解中，而仅损失一个 $(1+\epsilon)^2$ 的因子。

▶ **引理 13**。 有 $\ell_{i^*} \geq \ell_Y$。

证明。 由于 $\mathrm{OPT}'_Y \cap V'_{\ell_Y} \neq \emptyset$，$\left|\mathrm{OPT}' \cap V_{\ell_Y}\right| > 1/\epsilon$ 因此 OPT'_X 包含所有 $1/\epsilon$ 来自 $V^{1/\epsilon}_{\ell_Y}$ 的物品。因此，$p_X \geq \frac{1}{\epsilon} \cdot (1+\epsilon)^{\ell_Y}$。由上可得 $f() = y_{i^*}$ 以及 $f() \geq p_X$。根据 ℓ_{i^*}，

$$(1+\epsilon)^{\ell_{i^*}} = \lceil \epsilon \cdot y_{i^*} \rceil_{1+\epsilon} = \lceil \epsilon \cdot f(w_X) \rceil_{1+\epsilon} \geq \lceil \epsilon \cdot p_X \rceil_{1+\epsilon} \geq \left\lceil \epsilon \cdot \frac{1}{\epsilon}(1+\epsilon)^{\ell_Y} \right\rceil_{1+\epsilon} = (1+\epsilon)^{\ell_Y}.$$

这意味着 $\ell_{i^*} \geq \ell_Y$。 ◀

接下来，考虑我们通过查询得到的分数背包解。注意，该解的利润至少与 OPT'_Y 的利润一样大（因为分数背包是 0-1 背包的松弛）。此外，该分数解至多有一个物品以分数形式使用，且该物品来自 $U_{\ell_{i^*}}$，其价值至多为 $(1+\epsilon)^{\ell_{i^*}} = \lceil \epsilon \cdot y_{i^*} \rceil_{1+\epsilon} \leq (1+\epsilon)\epsilon \cdot y_{i^*}$。因此，我们可以将该物品计入 OPT'_X，最多损失一个 $(1+\epsilon)^2$ 的因子。

由此可知，解 $y_{i^*} + y'_{i^*}$ 是 OPT 的 $(1+\epsilon)^{O(1)}$-近似。结合之前的运行时间分析，我们得到定理 1。

4 技术概述

现在，我们概述两种使动态规划适用于我们框架的技术。我们将简要讨论如何单调化 k-平衡划分的动态规划，以及如何反转同时源定位的动态规划。受限于篇幅，我们只展示算法的部分内容，并且只考虑特殊情形。更具体地说，对于这两个问题，我们都考虑输入图为二叉树的特殊情形。在附录中，我们将证明这些结果可以推广到一般图。

4.1 单调化 Feldmann 和 Foschini 的动态规划

我们首先考虑 k-平衡图划分问题。回顾一下，该问题的输入是一个图 $G = (V, E, \mathrm{cap})$，其中 $\mathrm{cap}: E \to W_\infty$ 是边上的权重函数，且给定一个整数 k。正如引言中所讨论的，我们假设可以将划分大小放宽 $(1+\epsilon)$ 倍，我们的目标是找到顶点的一个划分 $V_1, \ldots\ldots, V_k$，使得对所有 i 都有 $|V_i| \leq \lceil (1+\epsilon)\,|V|/k \rceil$，并使 割$(V_1, \ldots\ldots, V_k) := \sum_{i=1}^{k} \sum_{\{u,\, v\} \in E \cap (V_i \times (V \setminus V_i))} \mathrm{cap}(u, v)$ 最小。

为了便于阐述，这里我们仅考虑 G 为一棵二叉树 的特殊情况；在附录 D.4 中，我们将说明如何去掉这一假设。

18 单调动态规划的动态维护及其应用

接下来，我们将介绍一种行具有单调性的动态规划，并说明如何利用单调分段常数函数高效地对这些解向量执行操作。我们的动态规划与 Feldmann 和 Foschini 的动态规划有关 [34]，后者并非单调，因此，我们的动态规划可以看作是对 Feldmann 和 Foschini 的动态规划进行单调化。我们相信，未来我们的动态规划单调化技术还将有更多应用。

动态规划的高层次描述。我们的动态规划从树的叶节点开始，自底向上计算。对于每个顶点 v，我们将计算一个代价最小的动态规划解，用以编码连接到父节点 p 的边是否被割断，以及以 v为根的子树 T_v 内部哪些边将被割断。注意，在我们的解中删除割边后，树将被分解为互不相交的连通分量，其中恰有一个连通分量包含 v的父节点 p。此外，我们还会记录删除割边后仍与父节点 p （因此也与T_v外部）连通的顶点数量。我们假设，在计算顶点 v的动态规划单元时，可以获取其两个子节点对应的解。

更具体地说，当我们已经计算出子树 T_v的一个解时，也就是说，我们知道要删除哪些与该子树中节点相连的边（注意，通向 v 父节点的边与 T_v 相连，因此我们将其视为该解的一部分），便会在动态规划表中存储以下信息。首先，存储其代价，即与 T_v 中的顶点相连且被割断的所有边的容量总和。如上所述，我们还希望存储与 v 的父节点连通的顶点数量，以及 T_v内部连通分量的大小。不过，这里有两个难点：（1）我们无法准确存储与根节点连通的顶点数量，因为这会导致动态规划表过大。因此，我们存储这样一种代价最低的解：其中至多有给定数量个顶点仍与 v的父节点连通。正如我们将看到的，这种方法会产生单调递减函数，并使动态规划表能够得到非常高效的计算。（2）我们以隐式方式存储删除割边后形成且完全位于 T_v内部的所有连通分量的大小。和前面一样，若要精确存储这些大小，动态规划表就会非常庞大，因此我们使用签名这一概念对其进行简洁表示。这些签名将帮助我们非常高效地刻画 T_v 内部连通分量的大小。

签名。我们将包含至少 $\epsilon\mathsf{d}|V|/k\mathsf{e}$ 个顶点的连通分量称为 T_v 大，否则称其为小。令 $t=\mathsf{dlog}_{1+\epsilon}(1/\epsilon)\mathsf{e}+1$，并令 $M=\mathsf{d}k/\epsilon\mathsf{e}+1$。签名是一个向量 $g=(g_0,\ldots,g_{t-1})\in[M-1]^t$。注意，每个 P_i 都是介于 0 和 $M-1$ 之间的整数，因此共有 $M^t=(k/\epsilon)^{O(\epsilon^{-1}\log(1/\epsilon))}$ 个不同的签名。直观来说，g 中的一个条目 P_i 大致表示，在我们考虑的动态规划解中，大小为 $(1+\epsilon)^i\cdot\epsilon\mathsf{d}|V|/k\mathsf{e}$ 的连通分量有多少个。由于篇幅所限，形式化定义请参见附录。

对于 $x\in\mathbb{N}$，我们令 $e(x)\in[M-1]^t$ 表示一个包含 x 个顶点的单个连通分量的签名。更准确地说，我们将 $e(x)$ 设为如下向量：当 $j=\arg\min\{j\in\mathbb{N}\colon x\le(1+\epsilon)^j\cdot\epsilon\lceil|V|/k\rceil\}$ 时，该向量的值为 $e(x)_j=1$；否则取 $e(x)_j=0$。如果 $x<\epsilon\lceil|V|/k\rceil$，我们将 $e(x)$ 定义为 $=\vec{0}$。

形式化动态规划定义。现在我们对动态规划进行形式化描述。顶点 v 的动态规划表中的一个条目 $\mathrm{DP}(v,g,\mathrm{cut})\in W_\infty$ 由签名 g、布尔值 cut 和$x\in[n]$索引。我们将元组 (v,g,cut) 视为动态规划表的行 I，并将 x 视为列；我们把每一行与一个函数 $\mathrm{DP}(v,g,\mathrm{cut},\cdot)\colon[n]\to W_\infty$相关联。注意，我们的动态规划有 $|V|\cdot M^t\cdot 2=(k/\epsilon)^{O(\epsilon^{-1}\log(1/\epsilon))}\cdot n$ 行。此外，它有 n 列；稍后，尽管 x 只取离散值，我们仍允许 x 取 $[0,\infty)$ 中的值。

条目 $\mathrm{DP}(v,\ g,\ \text{割}\ x)$ 表示切断与该子树相邻的边所需的最优代价

子树 T_v（包括可能切断与 v 的父节点相连的边所需的代价）。我们将 T_v 中在删除割边后仍与 v 的父节点连通的顶点集合称为根连通分量。我们对 DP(v, g, 割 x) 施加以下条件：

- 删除割边后，根连通分量 $U \subseteq T_v$ 至多包含 x 个顶点，即 $|U| \leq x$。如果将 割 设为 true，则 v 与其父节点之间的边会被割断；否则，该边会保留。在 T_v 内部，删除割边后与 v 的父节点不连通的顶点会形成与签名 g 一致的连通分量。接下来，我们观察到，如果固定一个顶点 v、一个签名 g 以及 割 的值，那么所得函数 DP(v, g, 割, ·) 关于 x 单调递减。

▶ **观察 14**。设 $v \in V$、$g \in [M-1]^t$ 是一个签名，且 *cut* $\in$ {true,false}。那么函数 动态规划$(v, g,$ *cut*$, \cdot): [0, \infty) \to \mathbb{R}_+$ 单调递减。

证明。根据定义，DP($v, g,$ cut x)存储根连通分量中顶点数至多为 x 的最优解的代价。由于 $x \leq x'$，解 DP($v, g,$ cut x)也是 DP($v, g,$ cut x')的一个可行解。因此，DP($v, g,$ cut, ·)是单调递减的。◀

与 Feldmann 和 Foschini 的动态规划比较。 将我们的动态规划与 Feldmann 和 Foschini 的动态规划进行比较时 [34]，一个关键区别是，在我们的动态规划中，x 表示根连通分量中顶点数的上界 。此前，Feldmann 和 Foschini 考虑的是恰好 含有 x 个顶点的根连通分量。这就是他们的动态规划非单调 的原因，也因此可以将我们的动态规划视为对 [34]中动态规划的单调化 。不过，我们还将动态规划推广到了带顶点权重的情形；正如我们将在下文看到的，我们用于近似计算动态规划的部分算法相当复杂。

4.1.1 计算动态规划

下面介绍一下用于计算动态规划的算法大致是什么样的。我们首先说明如何计算树中顶点 v 的精确动态规划解 DP(v,·,·,·)，其中 v 的父节点是 p，子节点是 v_l 和 v_r，并且它分别通过边 e_p、e_l 和 e_r 与它们相连。

计算动态规划需要进行若干分类讨论；这里我们只考虑这样的情况： v 是树的内部节点，并且我们不割断边 e_l 和 e_r。其他情况见附录。

计算由 DP(v,·,·,·) 给出的 DP 行时，我们只需要访问 DP 行 DP(v_l,·,·,·) 和 DP(v_r,·,·,·)。这意味着动态规划的依赖树是一棵树，其高度与输入图 G 的高度相同（回想一下，这里我们假设 G 是一棵二叉树）。请注意，树的高度也决定了可达节点数量的一个上界。

精确计算。 我们从精确计算开始。在这里，我们可以遍历所有值 $x \in [n]$ 和 $g \in [M-1]^t$，以计算动态规划(v,·,·,·)。因此，我们将 x 和 g 的值作为算法的输入。

由于我们假设不割断边 e_l 和 e_r，因此必须为 T_{v_l} 和 T_{v_r}选择子解，其中每个子解由上界 x_l（分别为 x_r）及其签名 g_l（分别为 g_r）来刻画。

首先，假设我们割去边 e_p。如果用 x_l 和 x_r 表示子解中根连通分量的顶点数，那么顶点 v 之后将包含在一个大小为 $x_l + x_r + 1$ 的连通分量中。因此，我们可以将这些子解组合起来，得到一个针对

签名 g 的解，只要 $g_l + g_r + e(x_l + x_r + 1) = g$。因此，我们对每个 $x \in [0, \infty)$ 设定

$$\text{动态规划}_B(v, g, \text{true}, x) = \text{容量}(v, p) + \min_{x_l, x_r, g_l+g_r=g-e(x_l+x_r+1)} \text{动态规划}(v_l, g_l, \text{false}, x_l) + \text{动态规划}(v_r, g_r, \text{false}, x_r).$$

其次，假设我们不切割 e_p。同样，我们必须为所有签名 g 以及所有 $x \in [0, 1)$ 设置动态规划$_B(v, g, \text{false}, x) = \infty$，因为 v 可以到达 p。对于 $x \geq 1$，我们必须选择 x_l和 x_r，使它们的和为 $x - 1$，这保证了最多只有 x 个顶点可以到达父节点 p。因此，我们对所有 $x \in [1, \infty)$ 进行设置

$$\text{DP}_B(v, g, \text{false}, x) = \min_{g_l+g_r=g, x_l+x_r=x-1} \text{DP}(v_l, g_l, \text{false}, x_l) + \text{DP}(v_r, g_r, \text{false}, x_r).$$

在这里，我们可以通过穷举所有 $O(M^t n^2)$ 种可能性，来处理上述 min- 操作。

近似计算。现在我们来考虑近似计算。我们用 ADP 表示近似动态规划解。假设我们已经计算出子节点的解 ADP($v_l, g,$ 割, ·) 和 ADP($v_r, g,$ 割, ·)，并使用第 2 节中的数据结构存储了它们。我们将维持这样一个不变量：这些函数各自最多有 $p = O(\log_{1+\delta}(W))$ 个分段；为此，我们会在每一步结束时对解进行舍入，即使用引理 6 中的舍入过程设置 ADP($v, g,$ 割, ·) $=$ dADP($v, g,$ 割 , ·)$e_{1+\delta}$。这将确保以下两个性质：(1) 根据引理 6，函数 ADP($v, g,$ 割 , ·) 的分段数不会超过 $O(\log_{1+\delta}(W))$。因此，我们可以非常高效地执行所有操作。(2) 对于树根处的函数，近似误差至多为 $(1 + \delta)^h$，其中 h 是树的高度。通过选取$\delta = O(\epsilon/h)$，我们将在根节点得到一个 $(1 + \epsilon)$-近似解。现在我们继续说明计算过程。

如果我们不割断连接到 v父节点的边，就按照上述精确动态规划中的类似方法继续。我们首先对所有 $x \in [0, 1$，设置 ADP$_B(v, g, \text{false}, x) = \infty$。接下来，对于 $x \in [1, \infty$，我们希望设置

$$\begin{aligned} \text{ADP}_B(v, g, \text{false}, x) &= \min_{g_l+g_r=g, x_l+x_r=x-1} \text{ADP}(v_l, g_l, \text{false}, x_l) + \text{ADP}(v_r, g_r, \text{false}, x_r) \quad (3) \\ &= \min_{g_l+g_r=g} \min_{x_l+x_r=x-1} \text{ADP}(v_l, g_l, \text{false}, x_l) + \text{ADP}(v_r, g_r, \text{false}, x_r). \quad (4) \end{aligned}$$

注意，对于固定的 g_l 和 g_r，第二行中的内层最小值运算描述了一个（min $+$）卷积，这是由约束 $x_l + x_r = x - 1$导致的。因此，在内层最小值运算中，我们计算卷积 ADP($v_l, g_l,$ false, ·) $\oplus$ ADP($v_r, g_r,$ false, ·)，并通过引理 6 中的平移操作将结果位移 1（对于 $x \in [0, 1,$ 我们设置 ADP$_B(v, g,$ false $x) = \infty$）。根据引理 7，计算该卷积需要时间 $O(p^2 \log p)$。为了计算公式（4）中的外层最小值，我们使用引理 6 遍历所有 $g_l \in [M - 1]^t$，因此需要对至多有p^2个分段的分段常数函数执行 $O(M^t)$ 次最小值计算。根据引理 21，这需要时间 $O(M^t p^2 \log(M^t p^2))$。根据引理 22，ADP$_B(v, g,$ false, ·) 是单调递减函数，因为它是两个单调递减函数的卷积的最小值。

如果我们割断通向 v的父节点的边，那么对于所有 $x \in [0, \infty)$，我们希望设置

$$\text{ADP}_B(v, g, \text{true}, x) = \text{cap}(v, p) + \min_{x_l, x_r, g_l+g_r=g-e(x_l+x_r+1)} \text{ADP}(v_l, g_l, \text{false}, x_l) + \text{ADP}(v_r, g_r, \text{false}, x_r).$$

注意，此处需要谨慎，因为 g_l和 g_r的值域取决于 $x_l + x_r$的选择。由于 $x_l + x_r$有 $\Omega(n)$ 个可能的取值，我们无法遍历所有 $x_l + x_r$的取值，

$x_l + x_r$ 可以取的值。相反，我们将利用 ADP(v_l, g_l，false，·) 和 ADP(v_r, g_r，false,·) 的单调性，证明只需考虑 $O(\log(k/\epsilon)/\epsilon)$ 个不同的数对（x_l, x_r）。

首先，注意到我们可以假设 $x_l \le |T_{v_l}|$ 和 $x_r \le |T_{v_r}|$：进一步增大根连通分量中顶点数量的上界，意味着根连通分量包含子树内的 所有 顶点，这是不可能的。因此，$x_l + x_r + 1 \in [1, n]$。

其次，我们将区间 $[1, n]$ 划分为 $O(\log(k/\epsilon)/\epsilon)$ 个区间。对于所有 $j = 1, \dots, \log_{1+\epsilon}(k/\epsilon)$，我们有区间$I_j = (\xi_{j-1}, \xi_j]$，其中 $\xi_j = (1+\epsilon)^j \epsilon \mathrm{d}n/k\mathrm{e}$。此外，我们加入一个“区间” $I_0 := [\epsilon \mathrm{d}n/k\mathrm{e}\ \epsilon \mathrm{d}n/k\mathrm{e}]$ 以及区间 $I_{-1} := [1, \epsilon \mathrm{d}n/k\mathrm{e})$。我们设置 $\xi_0 = \epsilon \mathrm{d}n/k\mathrm{e}$，并将 ξ_{-1} 设置为小于 $\epsilon \mathrm{d}n/k\mathrm{e}$ 的最大整数。注意，对于所有 $j \ge -1$和 $x \in I_j$，我们都有 $e(x) = e(\xi_j)$，即，$e(x)$ 的值在区间I_j内不会发生变化。下文中，这一性质将使我们能够分别处理关于 $x_l + x_r$ 和关于 $g_l + g_r$的条件。

现在我们可以将上述表达式改写为

$$\mathrm{ADP}_B(v, g, \mathrm{true}, x) = \mathrm{cap}(v, p) + \min_j \min_{x_l + x_r + 1 \in I_j} \min_{g_l + g_r = g - e(\xi_j)} \mathrm{ADP}(v_l, g_l, \mathrm{false}, x_l) + \mathrm{ADP}(v_r, g_r, \mathrm{false}, x_r).$$

第三，请注意，现在这两个最小值操作只取决于 j 的选择，而重要的是，对 g_l 和 g_r 取最小值不再取决于 $x_l + x_r$ 的选择。因此，我们可以交换这两个最小值操作的顺序。此外，由于 $\mathrm{ADP}_B(v,\ g,$ false $x)$ 关于 x 单调递减，我们可以将 x_l 和 x_r 的选择限制为使 $x_l + x_r + 1$ 成为相应区间 I_j 中的最大值，即 $x_l + x_r + 1 = \xi_j$。因此，

$\mathrm{ADP}_B(v, g,$真 $x) =$
容量$(v,\ p)$ +最小值$_j$最小值$_{g_l+g_r=g-e(\xi_j)}$最小值$_{x_l+x_r+1=\xi_j}$ADP$(v_l,\ g_l,$ false $x_l)$ +ADP$(v_r,\ g_r,$ false $\xi_j - x_l - 1)$。

接下来，我们说明如何高效计算上述表达式。首先，我们来说明如何高效计算上述表达式的内层最小值运算。我们注意到，由于约束中的求和上限为 ξ_i，这是一个常数（而不是变量 x），因此该最小值运算不是卷积。现在回想一下，根据我们的不变量，ADP($v_l, g_l,$ false, ·) 和 ADP($v_r, g_r,$ false, ·) 是具有 $O(p)$ 个分段的分段常数函数。由于 $x_l, x_r \ge 0$，这意味着对于 x_l 和 x_r，只有 $O(p^2)$ 种选择能使 $x_l, x_r \in I_j$，并且在 ADP($v_l, g_l,$ false x_l) 或 ADP($v_r, g_r,$ false x_r) 中有一个新分段开始。因此，我们可以遍历所有这些数对 (x_l, x_r)，并计算 ADP($v_l, g_l,$ false x_l) +ADP($v_r, g_r,$ false x_r)，其中 $x_r = \xi_j - x_l - 1$。由此，我们可以在 $O(p^2 \log p)$ 时间内计算内层最小值运算。注意，由于该最小值运算考虑了超常数个项，这个动态规划不是良好态的（它违反了定义 8 的性质 (4b)）。因此，在分析中我们将采用 C.2 节中更一般的概念。

接下来，我们只需遍历 j 和 g_l的所有取值，按照上述方法设置 $g_r = g - e(\xi_j) - g_l$，即可通过 $O(M^t \cdot \log(k/\epsilon)/\epsilon)$ 次迭代计算出最外层的两个最小值操作。因此，运行时间为 $O(M^t p^2 \log p \cdot \log(k/\epsilon)/\epsilon)$。

最后，我们注意到，由于 $\mathrm{ADP}_B(v, g,$true $x)$ 与 x 无关，因此它是一个常数。因此，$\mathrm{ADP}_B(v, g,$ true $x)$ 是一个仅含单个区间的分段常数函数，并且单调递减。

舍入步骤。如前所述，在计算出解 $\mathrm{ADP}_B(v, g,$ false, ·) 和 $\mathrm{ADP}_B(v, g,$ true, ·) 后，我们还会通过设置 $\mathrm{ADP}_B(v, g,$ cut, ·) $= \mathrm{dADP}_B(v, g,$ cut, ·$)\mathrm{e}_{1+\delta}$for cut $\in$ {true, false} 来对解进行舍入，以确保结果函数只有 $p = O(\log_{1+\delta}(W))$ 个分段。请注意，这是我们执行的唯一近似操作，以上所有其他操作都是精确的。

4.2 反转 Andreev 等人的动态规划

现在我们简要介绍用于同时源定位的动态规划。回顾一下，在这个问题中，输入由一个带容量函数 cap: $E \to W_\infty$ 和需求函数 d: $V \to W_\infty$ 的无向图 $G = (V, E,$ 容量 $d)$ 构成。目标是选取一个最小集合$S \subseteq V$的源点，使其能够同时满足所有顶点需求。更具体地说，如果存在一条从 S 中的顶点出发的流，能够向所有顶点 $v \in V$ 提供需求 $d(v)$，且不违反边上的容量约束，那么源点集合 S 就是可行 的。目标是找到一个大小最小的可行源点集合。

这里，我们仍假设特殊情形 G 是一棵二叉树；我们将在附录 E.3 中说明如何去掉这一假设。

动态规划定义。给定一个顶点 v 和一个值 $x \in \mathbb{R}$，我们令 DP(v, x) 表示需要在子树 T_v 中放置的源点最小数量，使得当 v 从其父节点接收至多 x 的流时，能够满足 T_v 中的所有需求。注意，x 可以取正值或负值：对于 $x \geq 0$，这对应于流从 v 的父节点流入 T_v 的情形；对于 $x < 0$，这对应于流从 T_v 流向 v 的父节点的情形。我们还遵循这样的约定：当 v 从其父节点接收 x 的流时，如果无法满足 T_v 中的需求，则令 DP$(v, x) = \infty$。

注意，该动态规划的行是$\mathcal{I} = V$，列是$\mathcal{J} = \mathbb{R}$。此外，DP$(v,\cdot)$是单调递减的，因为对于 $x < x'$，任何满足以下条件的解：T_v 从其父节点接收的流不超过 x，在 T_v 从其父节点接收的流不超过 x' 时仍然可行，其中父节点为 v 和 v。这满足定义8的性质（1）。

逆动态规划。有趣的是，我们的动态规划与 Andreev 等人提出的动态规划 [4]密切相关。他们定义了一个函数 $f(v, i)$，对于给定的顶点 v和整数 $i \in \mathbb{N}$，该函数表示：若需要满足 T_v 中的所有需求，且可以在子树 T_v中放置 i 个源点，则 v 需要从其父节点接收的最小流量 。与上文类似，当 T_v 中的需求只能通过从 v 的父节点接收流来满足时，$f(v, i)$取正值；当 T_v 中的需求已由子树 T_v 中的源点满足，且 v 可以向其父节点发送流时，该函数取负值。

现在注意到，我们的动态规划本质上可以看作是 $f(v,\ i)$ 的“逆”。更正式地说，注意到 DP$(v,\ x) = f^{-1}(v,\ x) := \min\{i\colon\ f(v,\ i) \leq x\}$。

我们为动态规划选择逆向表述的原因如下。为了确保我们的算法高效，必须保证单调分段常数函数只有少量分段。一种自然的做法是使用舍入。然而，由于 f 的函数值既有正值也有负值，我们不确定应该如何进行舍入。例如，为了用少量分段表示 f，我们必须对 $[-1, 1]$ 中的函数值和 $[-W, W] \setminus [-1, 1]$ 中的函数值采用不同的舍入机制，其中 W 是最大边容量：确实，如果将 f 的值舍入到 $(1+\delta)^j$ 的幂，那么 $[-W, W] \setminus [-1, 1]$ 中只有 $O(\log_{1+\delta}(W))$ 个函数值，但 $[-1, 1]$ 中却有无穷多个函数值。同样，如果将值舍入到 δ 的倍数，那么 $[-1, 1]$ 中只有 $O(1/\delta)$ 个函数值，但这会导致 $[-W, W]$ 中有 $O(W/\delta)$ 个函数值 $\setminus [-1, 1]$。在这两种情况下，我们的函数都会有过多分段，因此必须选择一种能在这两种情况之间取得平衡的舍入函数。此外，我们还必须进行分析，证明这种“更复杂”的舍入函数不会引入过大的误差。

在我们的动态规划中，我们通过将负数移入函数 DP$(v,\ \cdot)$ 的定义域来绕过这些问题：$\mathbb{R} \to [n+1]$。这样，陪域中就只有非负数，因此可以直接应用标准舍入函数$\lceil\cdot\rceil_{1+\delta}$。这还带来了一个积极的副作用：我们的运行时间中不再出现 polylog(W) 因子，而是

只出现 polylog(n) 因子，因为我们的陪域变成了 $[n+1]$，而不是某个可能很大的区间 $[-W, W]$。我们认为，考虑逆动态规划这一技巧在未来有助于计算取正和负 值的动态规划的近似解。

由于空间有限， 我们在附录 E 中给出计算动态规划的详细内容 n 附录 E.

参考文献

1 Amir Abboud、Vincent Cohen-Addad 和 Philip N Klein。平面图问题在 P 中的新困难性结果及稀疏割算法。载于 STOC，第 996–1009 页，2020 年。 2 Alok Aggarwal、Maria M. Klawe、Shlomo Moran、Peter W. Shor 和 Robert E. Wilber。矩阵搜索算法的几何应用。Algorithmica，2：195–208，1987 年。 3 Greg Aloupis、Thomas Fevens、Stefan Langerman、Tomomi Matsui、Antonio Mesa、Yurai Nuñez、David Rappaport 和 Godfried Toussaint。计算旋律相似性几何度量的算法。Computer Music Journal，30(3)：67–76，2006 年。 4 Konstantin Andreev、Charles Garrod、Daniel Golovin、Bruce M. Maggs 和 Adam Meyerson。同时源定位。ACM Trans. Algorithms，6(1)：16:1–16:17，2009 年。 5 Konstantin Andreev 和 Harald Räcke。平衡图划分。Theory of Computing Systems，39(6)：929–939，2006 年。 6 Ali Aouad 和 Danny Segev。增量背包问题的近似动态规划方法。Operations Research，2022 年。 7 Kouji Arata、Satoru Iwata、Kazuhisa Makino 和 Satoru Fujishige。在无向网络中定位源点以满足流需求。Journal of Algorithms，42(1)：54–68，2002 年。 8 Yoan José Pinzón Ardila、Raphaël Clifford 和 Manal Mohamed。用于节奏相似性度量的项链交换问题。载于 International Symposium on String Processing and Information Retrieval，第 234–245 页。Springer，2005 年。 9 Kyriakos Axiotis 和 Christos Tzamos。容量约束动态规划：更快的背包和图算法。载于 ICALP，第 19:1–19:13 页，2019 年。 10 Arturs Backurs、Piotr Indyk 和 Ludwig Schmidt。近线性时间内更优的树稀疏性近似。载于 SODA，第 2215–2229 页，2017 年。 11 MohammadHossein Bateni、MohammadTaghi Hajiaghayi、Saeed Seddighin 和 Cliff Stein。通过卷积与预测实现的快速背包算法。载于 STOC，第 1269–1282 页，2018 年。 12 Wolfgang W. Bein、Mordecai J. Golin、Lawrence L. Larmore 和 Yan Zhang。Knuth–Yao 四边形不等式加速是全单调性的推论。ACM Trans. Algorithms，6(1)：17:1–17:22，2009 年。 13 Marcin Bienkowski、Miroslaw Korzeniowski 和 Harald Räcke。构造无感知路由方案的实用算法。载于 SPAA，第 24–33 页，2003 年。 14 David Bremner、Timothy M. Chan、Erik D. Demaine、Jeff Erickson、Ferran Hurtado、John Iacono、Stefan Langerman、Mihai Pătraşcu 和 Perouz Taslakian。项链、卷积与 X+Y。Algorithmica，69(2)：294–314，2014 年。 15 Karl Bringmann 和 Alejandro Cassis。通过有界单调最小加卷积实现更快的背包算法。载于 ICALP，第 229 卷，第 31:1–31:21 页，2022 年。 16 Karl Bringmann、Fabrizio Grandoni、Barna Saha 和 Virginia Vassilevska Williams。通过快速有界差分最小加乘积实现语言编辑距离和 RNA 折叠的真正次立方算法。SIAM J. Comput.，48(2)：481–512，2019 年。 17 Karl Bringmann、Marvin Künnemann 和 Karol Wegrzycki。无缩放近似 APSP：近似最小加与精确最小最大之间的等价性。载于 STOC，第 943–954 页。ACM，2019 年。 18 Aydin Buluç、Henning Meyerhenke、Ilya Safro、Peter Sanders 和 Christian Schulz。图划分的最新进展。载于 Algorithm Engineering - Selected Results and Surveys，《计算机科学讲义》第 9220 卷，第 117–158 页，2016 年。

19 Rainer E. Burkard、Bettina Klinz 和 Rüdiger Rudolf。优化中蒙日性质的展望。Discret. Appl. Math.，70(2)：95–161，1996 年。 20 Michael Bussieck、Hannes Hassler、Gerhard J Woeginger 和 Uwe T Zimmermann。最大卷积问题的快速算法。Operations research letters，15(3)：133–141，1994 年。 21 Timothy M. Chan。0-1 背包的近似方案。载于 SOSA，第 61 卷，第 5:1–5:12 页，2018 年。 22 Timothy M. Chan。全单调矩阵中选择问题的近最优随机化算法。载于 SODA，第 1483–1495 页，2021 年。 23 Timothy M. Chan 和 Moshe Lewenstein。通过加法组合学解决聚类整数 3SUM。载于 STOC，第 31–40 页，2015 年。 24 Shucheng Chi、Ran Duan、Tianle Xie 和 Tianyi Zhang。单调实例上更快的最小加乘积。载于 STOC，第 1529–1542 页，2022 年。 25 Justin Colannino、Mirela Damian、Ferran Hurtado、John Iacono、Henk Meijer、Suneeta Ramaswami 和 Godfried Toussaint。限制支架分配问题的一个 o(n log n) 时间算法。《计算生物学杂志》，13(4)：979–989，2006 年。 26 Spencer Compton、Slobodan Mitrovic 和 Ronitt Rubinfeld。新的划分技术与更快的近似区间调度算法。CoRR，abs/2012.15002，2020 年。URL:https://arxiv.org/abs/2012.15002。 27 Marek Cygan、Marcin Mucha、Karol Wegrzycki 和 Michal Wlodarczyk。关于与 (min，+)-卷积等价的问题。ACM Trans. Algorithms，15(1)：14:1–14:25，2019 年。 28 Yihe Dong、Piotr Indyk、Ilya P. Razenshteyn 和 Tal Wagner。学习用于最近邻搜索的空间划分。载于 ICLR，2020 年。 29 Franziska Eberle、Nicole Megow、Lukas Nölke、Bertrand Simon 和 Andreas Wiese。具有多对数更新时间的背包问题全动态算法。载于 FSTTCS，第 213 卷，第 18:1–18:17 页，2021 年。 30 Eduard Eiben、Daniel Lokshtanov 和 Amer E Mouawad。通过卷积对有界树宽图进行二分划分。载于 ESA 2019。Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik，2019 年。 31 David Eppstein、Zvi Galil 和 Raffaele Giancarlo。加速动态规划。载于 FOCS，第 488–496 页。IEEE计算机学会，1988 年。 32 Guy Even、Joseph Naor、Satish Rao 和 Baruch Schieber。快速近似图划分算法。SIAM J. Comput.，28(6)：2187–2214，1999 年。 33 Uriel Feige 和 Robert Krauthgamer。最小二分划分的多对数近似。SIAM J. Comput.，31(4)：1090–1118，2002 年。 34 Andreas Emil Feldmann 和 Luca Foschini。树的平衡划分及其应用。Algorithmica，71(2)：354–376，2015 年。 35 Zvi Galil 和 Kunsoo Park。具有凸性、凹性和稀疏性的动态规划。Theor. Comput. Sci.，92(1)：49–76，1992 年。 36 Gramoz Goranci、Harald Räcke、Thatchaphol Saranurak 和 Zihan Tan。扩展图层次结构及其在动态图算法中的应用。载于 SODA，第 2212–2228 页，2021 年。 37 Chris Harrelson、Kirsten Hildrum 和 Satish Rao。用于最小化拥塞的多项式时间树分解。载于 SPAA，第 34–43 页，2003 年。 38 Monika Henzinger、Stefan Neumann 和 Andreas Wiese。区间、超立方体和超矩形的动态近似最大独立集。载于 SoCG，《LIPIcs》第 164 卷，第 51:1–51:14 页，2020 年。 39 Dorit S Hochbaum 和 David B Shmoys。使用对偶近似算法解决调度问题：理论与实践结果。《ACM杂志》，34(1)：144–162，1987 年。 40 Yiding Hua、Rasmus Kyng、Maximilian Probst Gutenberg 和 Zihang Wu。通过稀疏割维护扩展图分解。CoRR，abs/2204.02519，2022 年。 41 Piotr Indyk、Fabian Kuhn 和 Anca Muscholl。关于一维动态规划的细粒度复杂性。载于 ICALP 2017，第 80 卷，第 21 页，2017 年。

42 Klaus Jansen 和 Lars Rohwedder。《关于整数规划、差异度与卷积》。arXiv 预印本 arXiv:1803.04744，2018。43 Ce Jin。《一种改进的 0-1 背包全多项式时间近似方案》。载于 ICALP，第 132 卷，第 76:1–76:14 页，2019。44 George Karypis 和 Vipin Kumar。《Metis：用于划分非结构化图、划分网格以及计算稀疏矩阵减少填充的排序的软件包》。1997。45 Donald E. Knuth。《最优二叉搜索树》。Acta Informatica，1:14–25，1971。46 Soh Kumabe 和 Yuichi Yoshida。《动态规划的平均敏感性》。载于 SODA，第 1925–1961 页，2022。47 Eduardo S Laber、Wilfredo Bardales 和 Ferdinando Cicalese。《关于最大连续子和问题及 (min,+)-卷积的下界》。载于 ISIT，第 1807–1811 页。IEEE，2014。48 Gary L. Miller、Richard Peng、Russell Schwartz 和 Charalampos E. Tsourakakis。《使用半空间查询和多尺度 Monge 分解的近似动态规划》。载于 SODA，第 1675–1682 页，2011。49 Gaspard Monge。《关于挖方与填方理论的回忆录》。Mem. Math. Phys. Acad. Royale Sci.，第 666–704 页，1781。50 Marcin Mucha、Karol Węgrzycki 和 Michał Włodarczyk。《划分问题的次二次近似方案》。载于 SODA，第 70–88 页。SIAM，2019。51 Richard Peng。《在 O(mpolylog(n)) 时间内求解无向近似最大流》。载于 SODA，第 1862–1867 页，2016。52 Harald Räcke。《最小化一般网络中的拥塞》。载于 FOCS，第 43–52 页，2002。53 Harald Räcke 和 Chintan Shah。《改进树形割稀疏器的保证》。载于 ESA，第 774–785 页，2014。54 Harald Räcke、Chintan Shah 和 Hanjo Täubig。《近线性时间内计算基于割的层次分解》。载于 SODA，第 227–238 页，2014。55 Peter Sanders 和 Christian Schulz。《立足局部，着眼全局：高度平衡的图划分》。载于 SEA，第 7933 卷，第 164–175 页，2013。56 Hiroshi Tamura、Masakazu Sengoku、Shoji Shinoda 和 Takeo Abe。《无向流网络上的选址问题》。IEICE TRANSACTIONS (1976-1990)，73(12):1989–1993，1990。57 Hiroshi Tamura、Masakazu Sengoku、Shoji Shinoda 和 Takeo Abe。《流网络选址理论中的若干覆盖问题》。IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences，75(6):678–684，1992。58 Godfried Toussaint。《音乐节奏几何》。载于日本离散与计算几何会议，第 198–212 页。Springer，2004。59 Godfried T Toussaint 等。《节奏相似性度量的比较》。载于 ISMIR，2004。60 Nithin Varma 和 Yuichi Yoshida。《图算法的平均敏感性》。载于 SODA，第 684–703 页，2021。61 R. Ryan Williams。《通过电路复杂性实现更快的全点对最短路径算法》。SIAM J. Comput.，47(5):1965–1985，2018。62 Virginia Vassilevska Williams。《算法与复杂性中的若干细粒度问题》。载于 ICM 会议论文集，第 3 卷，第 3431–3472 页。World Scientific，2018。63 Andrew Chi-Chih Yao。《概率计算：迈向复杂性的统一度量》。载于 FOCS，第 222–227 页，1977。64 F. Frances Yao。《利用四边形不等式实现高效动态规划》。载于 STOC，第 429–435 页，1980。

A 附录的组织结构

我们的附录安排如下：

- 在附录 B 中，我们讨论更多相关工作。附录 C 介绍预备知识。附录 D 展示我们关于 k-
- 平衡划分的结果。附录 E 展示我们关于同时源定位的结果。附录 F 展示了针对仅维护少
- 量解的算法所得的重构次数下界。附录 G 展示了我们对具有非单调性的函数所作的推广，
- 以及关于 ℓ_∞-项链的结果。附录 H 给出了缺失的证明。

B 更多相关工作

加速动态规划算法是一个经过深入研究的课题，数十年来一直备受关注 [2,12,19,22,31,35,45,48,49,64]。这一研究方向提出了若干条件，只要满足这些条件，就能更高效地求解相应的动态规划。这些条件包括 Monge 性质、全单调性、某些凸性和凹性性质，或 Knuth–Yao 四边形不等式；它们之间往往也存在关联。例如，众所周知，满足 Monge 性质的动态规划表也具有全单调性。该领域最受欢迎的方法之一是 SMAWK 算法 [2]；若动态规划表具有全单调性，该算法的运行时间相对于动态规划表的列数为近线性时间。更具体地说，若对于动态规划表的每个子矩阵 A 及其中每一对相邻行 i 和 $i+1$，在 A 中，行 $i+1$ 的最小值所在列不小于行 i 的最小值所在列，则该动态规划表具有全单调性。

然而，这些条件与定义 8 中的条件截然不同，二者本质上不可比较。为便于说明，我们将简要论证全单调性与定义 8 之间的这一点；对于 Monge 性质和其他判据，也可以作出类似论证。一方面，全单调矩阵并不能保证动态规划表的各行单调。事实上，如果各行单调，那么找到最小值所在的列就轻而易举（根据各行是单调递增还是单调递减，它们总是在第一列或最后一列）。因此，全单调性并不蕴含定义 8 中的条件。另一方面，对于上述条件而言，各行的顺序至关重要：只要交换全单调动态规划表中的两行，就可能破坏全单调性。而在我们的情形中，只要依赖图具有良好的性质，动态规划表中的各行就可以任意排序。因此，我们的性质并不蕴含全单调性。这表明这些定义彼此不可比较。

近期，Varma 和 Yoshida [60] 以及 Kumabe 和 Yoshida [46] 研究了图算法和动态规划算法的敏感性。他们研究了从输入中随机删除一个元素时，这类算法的解会发生多大变化。对于包括背包问题在内的若干问题，他们证明这些算法的敏感性较小。然而，我们将在 F 节证明，当允许插入操作时，动态算法必须具有较高的重配置代价，否则就必须维护许多不同的解。

k平衡图划分问题在理论界 [5,32–34]受到了广泛关注。该问题在实践中也与实际应用密切相关 [18,28,44,55],，因为平衡图划分算法常被用作大规模数据分析的预处理步骤。对于 $k = 2$这一特殊情形，它对应于最小

二分问题，Feige 和 Krauthgamer [33] 提出了具有多对数近似比的多项式时间算法。对于 $k \geq 3$，Andreev 和 Räcke [5] 证明，除非 P $=$NP，否则不存在能达到有限近似比的多项式时间算法。他们还展示了如何在多项式时间内计算一个双准则（$O(\log^{1.5}(n)/\epsilon^2)$ $1+\epsilon$）近似解。Feldmann 和 Foschini [34] 提出了一种多项式时间双准则（$O(\log^{1.5}(n) \log\log n)$ $1+\epsilon$）近似算法，其优点是近似比不依赖于划分大小的参数 ϵ 。Even 等人 [32] 证明，可以在多项式时间内计算一个双准则（$O(\log n)$，2）近似解。

我们研究的同时源定位问题与 Tamura 等人提出的源定位问题密切相关。在该问题中，必须选出最少数量的源点，以便能够满足无向边容量图中的任意单个需求。Arata 等人证明了该问题是 NP 困难的，并针对统一顶点成本的变体提出了一种精确算法。在 Andreev 等人提出、且本文研究的同时源定位问题中，所有需求都必须同时得到满足。Andreev 等人为该问题提供了一种 $O(\log D)$-近似算法，其中D 是需求之和；他们还给出了该问题在一般图中的相匹配的困难性结果。当输入图为树时，他们还提出了一种精确多项式时间算法，并证明：当边容量允许以 $O(\log^2 n \log\log n)$ 倍的因子被违反时，该结果可以扩展到一般图，其中 n 是图中的顶点数量。

Chan [21] 指出，可以将 0-1 背包的解视为单调分段常数函数，并利用这一发现获得更快的算法。最近，Jin [43] 改进了这些结果，展示了如何在时间 $\tilde{O}(n+\epsilon^{-9/4})$ 内为包含 n 个物品的 0-1 背包计算 $(1+\epsilon)$-近似解。Bringmann 和 Cassis [15] 利用有界单调最小加卷积推导出更快的 0-1 背包精确算法。Aouad 和 Segev [6] 研究了增量背包问题：容量约束随时间增加，目标是找出能使平均利润最大化的嵌套物品子集。我们指出，这与我们的设定不同：我们的目标是实现高效的更新时间，而解可能随时间任意变化。

Toussaint [58],最初研究了 ℓ_1-项链对齐问题，其动机来自计算音乐理论和节奏相似性 [59]。Toussaint 关注的是珠子位于整数坐标上的情形。Ardila 等人 [8] 研究了二进制字符串上的这一问题。此外，还有关于实数轴上两个点集之间不同距离度量的研究，其中并非每个点都需要匹配 [25], ；也有研究探讨了当两段旋律表示为圆柱面上的闭合正交链时，如何计算它们的相似度 [3]。Bremner 等人 [14] 证明，使用快速傅里叶变换，可以在时间$O(n \log n)$内解决 ℓ_2-项链对齐问题，其中 n 为珠子数量。他们还证明，可以通过常数次 (min $+$)-操作解决 ℓ_∞-项链对齐问题，并为 ℓ_1-项链和 ℓ_∞-项链对齐问题给出了亚二次时间算法。

求解动态规划时常用的一个子程序是 (min $+$)-卷积；值得注意的是，这个子程序在我们所有的算法中也至关重要。(min $+$) 卷积的复杂性在文献中受到了广泛关注 [9–11, 14, 17, 20, 23, 24,27,41,42,47,50]。研究表明，通过将其归约到全点对最短路径 [14]，并使用 Williams 算法求解后者 [14]，可以将运行时间为 $O(n^2)$ 的朴素算法改进为运行时间 $n^2/2^{\Omega(\sqrt{\log n})}$ [14,61]。然而，迄今为止尚未发现运行时间为 $O(n^{2-\epsilon})$ 的算法，这促成了细粒度复杂性理论中的 $MinConv$ 困难猜想 [27, 41]。这一猜想尤其引人注目，因为它蕴含了其他猜想，例如 3-SUM 和全点对最短路径猜想，以及由此得到的数十个下界

（见 [27, 62]）。此外，还有许多基于 MinConv 猜想的条件下界，也已知若干 MinConv 等价问题，例如与背包问题或次可加序列相关的问题 [27, 41],，以及其他问题 [1, 10, 27, 30, 41, 42, 47, 50]。针对大权重的情形 [17]，以及针对有界差分的精确 (min +)-矩阵乘积 [16]，人们也改进了 (min +)-卷积的高效近似算法。

C 预备知识

我们先介绍一些将在本文其余部分中用到的预备知识。为提高可读性，我们将部分证明放在附录 H 中。我们用 $[m]$ 表示集合{0，1，.....，m}。

在本文中，我们将考虑具有 n 个顶点和 m 条边的输入图 $G = (V_G, E_G, \text{cap}_G)$，其中 $\text{cap}_G: E_G \to W_\infty \cup \{\infty\}$ 是一个权重函数，它为边 $e \in E_G$描述该边的容量。为简化记号，我们将 cap_G 扩展到所有顶点对，并定义

$$\text{cap}_G(x,y) = \begin{cases} \text{cap}_G(\{x,y\}) & \{x,y\} \in E_G \\ 0 & \text{otherwise.} \end{cases} .$$

此外，对于互不相交的集合 $A, B \subseteq V_G$，我们令 $\text{cap}_G(A, B) := \sum_{(a,\ b)\in A\times B} \text{cap}_G(a, b)$，且 $\text{cap}_G(A) := \text{cap}_G(A, V \setminus A)$。当上下文已明确图时，我们省略容量函数 cap 的下标 G。

设 (V_T, E_T, r) 是一棵有根树。对于顶点 $v \in V_T$，我们用 T_v 表示以 v 为根的子树，并称 v 的度为其子节点数。 h 的 T 高度是从根节点到叶节点的最长路径长度。

C.1 Räcke 树

无向图 $G = (V_G, E_G)$ 的一棵*Räcke* 树 [52]（或树割稀疏化器） $T = (V_T, E_T)$ 是一棵带权有根树，其叶节点对应于图 G 的顶点。对于顶点$v \in V_T$，我们用 $V_v \subseteq V_G$ 表示 T_v 中叶节点的集合。自然地， T 的一条边 $e = (u, v)$ 对应于 G 中的一个割，即由 G 中的集合 $V_u \cap V_v$ 所形成的割。树边 (u, v) 的容量 cap_T 设为该割的容量，即 $\text{cap}_G(V_u \cap V_v)$。

对于图 $H = (V_H, E_H)$ 和两个不相交子集 $A, B \subseteq V_H$，我们记作

$$\text{mincut}_H(A, B) := \min_{S\subseteq V_H: A\subseteq S,\ B\subseteq \bar{S}} \text{容量}_H(S)$$

表示将 A 和 B 分开的割的最小容量。根据 T 中边容量的定义，对于任意两个不相交子集 A, $B \in V_G$，我们有 $\text{mincut}_T(A, B) \geq \text{mincut}_G(A, B)$。为完整起见，我们在附录 H.5 中证明这一性质。

Räcke 树 T 的目标是近似 G 的割结构，即保证对于所有顶点的不相交子集 A, $B \subseteq V_G$,

$$\text{mincut}_G(A, B) \leq \text{mincut}_T(A, B) \leq q \cdot \text{mincut}_G(A, B) ,$$

当 $q \geq 1$ 取较小值时。参数 q 称为 Räcke 树的质量 。

在静态情形中，可以在近线性时间内计算出质量保证为多对数级的 Räcke 树 [51, 54]。如果允许更长的运行时间，则可以获得更好的质量 [13, 37, 53]。

▶ **定理 15** (Peng [51])。设 G 是一个具有 n 个顶点和m 条边的连通无向图。那么存在一种算法，可以为 G计算一棵高度为 $O(\log n)$ 、质量为 $O(\log^4 n)$ 的 ***Räcke*** 树，运行时间为 $\tilde{O}(m)$。

此外，近期人们对动态维护 Räcke 树产生了兴趣 [36,40]。这里，我们将采用 Goranci、Räcke、Saranurak 和 Tan 的一项结果，他们证明了可以用次多项式更新时间动态维护无权图的 Räcke 树。

▶**定理 16** (Goranci、Räcke、Saranurak 和 Tan [36])。设 G 是一个具有 n 个顶点、正在进行边插入和删除的无向无权图。存在一种确定性算法，其摊销更新时间为 $n^{o(1)}$ ，可为 G维护一棵 ***Räcke*** 树，其质量为 $n^{o(1)}$ ，高度为 $O(\log^{1/6} n)$。

C.2 良性 DP

我们引入一种比定义 8 中的条件更一般的 DP 条件，但它无法使我们获得类似定理 9 或 10 的结果。我们将考虑与第 2 节相同类型的动态规划表。

▶**定义 17.** 如果一个动态规划良态，则称其满足良好态动态规划的敏感性条件：假设 $\beta > 1$，且对于所有 $i' \in \mathrm{In}(i)$，我们得到一个 β-近似$\mathsf{ADP}(i', \cdot)$ ，它是$\mathsf{DP}(i', \cdot)$的近似（根据方程 (1)）。那么，在$\mathsf{ADP}(i', \cdot)$ 上应用 P_i，可得到$\mathsf{DP}(i, \cdot)$的一个 β-近似，即

$$\mathsf{DP}(i,\cdot) \le \mathcal{P}_i(\{\mathsf{ADP}(i',\cdot) \colon i' \in \mathrm{In}(i)\}) \le \beta \cdot \mathsf{DP}(i,\cdot).$$

˜我们还使用例程$\mathcal{P}_i$ 来计算 DP 行 ADP$(i,\cdot)$。同样，如果对于所有 i ，都有˜$\mathcal{P}_i(\{\mathsf{ADP}(i',\cdot)\colon \ i' \in \mathrm{In}(i)\})$是$\mathcal{P}_i(\{\mathsf{ADP}(i',\cdot)\colon \ i' \in \mathrm{In}(i)\})$ 的一个 α-近似，那么我们称 ADP$(1,\cdot)\ldots,$ADP$(n,\cdot)$ 是一个 α-近似动态规划解。

在依赖图中，我们将没有任何入边的顶点称为叶节点。顶点 u 的层级是从叶节点到 u 的最长路径长度。类似于定理 9 的证明，我们可以证明近似解 ADP$(i,\cdot)$ 和精确解 DP$(i,\cdot)$ 满足以下近似保证。

▶ **引理 18.** 设 i 是依赖图中层级为 ℓ 的一个顶点。那么，针对一个良性 ***DP*** 问题的 α-近似 ***ADP*** 解中的条目 $\mathsf{ADP}(i,\cdot)$满足

$$\mathsf{DP}(i,\cdot) \le \mathsf{ADP}(i,\cdot) \le \alpha^{\ell+1} \cdot \mathsf{DP}(i,\cdot).$$

接下来，假设我们所考虑的动态规划的依赖图是由一棵树按如下方式导出的。设 $T = (V_T,\ E_T,\ r)$ 是一棵根为 r、高度为 h 的有根树。我们假设一个顶点的子节点从左到右排列。与 T 相关联的依赖图就是 T 的一个有向副本，其中每条边都指向根节点。更准确地说，依赖图包含 V_T 中所有顶点的副本，并且对每个顶点 v（ r 除外），都添加一条指向其父节点 p 的边。显然，这些边构成一个有向无环图，其中最长路径至多包含 h 条边。以下引理总结了采用这种方法时近似动态规划解的性质。

▶ **引理 19**。考虑一棵高度为 h 的有根树 $T = (V_T, E_T, r)$。考虑一个良态动态规划，以及与上文所述依赖图对应的 ***ADP*** 解 $\mathsf{ADP}(i,\cdot)$ 。假设每个 P_i 都是 P_i 的 α-近似，且计算时间至多为 t。那么，$\mathsf{ADP}(r,\cdot)$ 是 $\mathsf{DP}(r,\cdot)$ 的 α^{h+1}-近似，且可在 $O(|V_T| \cdot t)$ 时间内计算。

30 单调动态规划的动态维护及其应用

本引理与良态动态规划的定义及良好动态规划对应的定理 9 之间的主要区别如下。应用定理 9 时，我们只需考虑函数有多少个分段，而无须费心推导计算函数上各项操作的运行时间上界（因为良好动态规划的附加条件蕴含了良好的运行时间上界）。而在此处，良态动态规划所需检查的条件较少（尤其是无须限制分段数或操作数），但我们必须自行进行运行时间分析。

之后，当我们考虑动态算法时，必须考虑底层树 T 因边插入与删除而发生变化（因此可能变成森林）的情形。在这种情况下，依赖图以及动态规划解 DP$(i,\ \cdot)$ 和 ADP$(i,\ \cdot)$ 也会随时间变化。以下引理说明，当顶点 i 受到边插入或删除的影响时，我们只需为依赖图中从 i 可达的顶点 j 重新计算解 DP$(j,\ \cdot)$ 和 ADP$(j,\ \cdot)$，并且这样的顶点至多有 h 个。

▶ **引理 20.** 考虑一棵高度为 h、正在进行边插入和删除的有根树 $T = (V_T, E_T, r)$。那么，每次插入或删除后，我们都可以在时间 $O(h\cdot t)$ 内重新计算一个具有与引理 *19* 相同保证的 *ADP* 解，其中 t 是计算函数 P_i 所需的时间~。

引理 6 已经给出了计算两个单调分段常数函数的最小值的方法。当最小值计算涉及两个以上的函数时，以下版本能提供更好的保证。

▶ **引理 21.** 设 $f_i\colon [0,t] \to W_\infty$，$i \in \{1,\dots,k\}$ 是分段常数函数，且它们要么全部单调递增，要么全部单调递减。则 $f_{\min}(x) := \min_i\{f_i(x)\}$ 可在 $O(\sum_i p_i \cdot \log(\sum_i p_i))$ 时间内计算，其中 p_i 表示函数 f_i 的分段数。

为完整起见，我们还指出以下众所周知的引理。

▶ **引理 22.** 设 $f_1, f_2\colon [0,t] \to W_\infty$，并假设 f_1 和 f_2 中有一个单调递减。那么 $f = f_1 \oplus f_2$ 单调递减。

D 平衡图划分

本节中，我们给出一个用于 k-平衡图划分问题的算法。在这个问题中，输入由一个图 $G = (V, E, \mathrm{cap})$ 以及一个整数 k 组成，其中 $\mathrm{cap}: E \to W_\infty$ 是定义在边上的权重函数。目标是找到顶点的一个划分 $V_1, \dots, V_k$，使得对所有 i 都有 $|V_i| \le \mathrm{d}|V|/k\mathrm{e}$，并使被该划分切开的边的权重最小化。更正式地说，我们要最小化 $\mathrm{cut}(V_1, \dots, V_k) := \sum_i \mathrm{cap}(V_i)$，其中 $\mathrm{cap}(V_i) = \sum_{\{u,v\}\in E\cap(V_i, V\setminus V_i)} \mathrm{cap}(u,v)$。

由于上述问题即使在树上也无法在任何因子 $n^{1-\epsilon}$ 下 ϵ进行 NP 困难的近似 [34], ，我们考虑双准则近似算法。给定一个加权图$G = (V,\ E,$ 容量$)$，如果划分 $V_1,\ \ldots\ldots,\ V_k$ 满足 V中的所有 i均有 $|V_i| \le \beta\lceil n/k\rceil$，且 割$(V_1,\ \ldots\ldots,\ V_k) \le \alpha \cdot$ 割(OPT)，则称其为 $(\alpha,\ \beta)$-近似解；其中，OPT $= (V_1^*,\ \ldots\ldots,\ V_k^*)$ 是满足所有 i均有 $|V_i^*| \le \lceil n/k\rceil$ 的最优解。

本节的第一个主要结果总结如下定理。我们使用记号 $O'(\cdot)$ 来省略 poly(log $n, k,$ log($1/\epsilon$), log log(W)) 中的因子。

▶ **定理 2**。 设 $\epsilon > 0$ 和 $k \in \mathbb{N}$。设 $G = (V, E,$ 容量) 是一个具有 n 个顶点和 m 条边、边权重取自 W_∞的无向加权图。那么对于 k-平衡划分问题，我们可以计算：

- 在时间 $(k/\epsilon)^{O(\log(1/\epsilon)/\epsilon)} \cdot O'(m \cdot \log^2(W)) + (k/\epsilon)^{O(1/\epsilon^2}$ 内的 $(O(\log^4 n)\ 1+\epsilon)$-近似。若 G 是一棵高度为 h 的树，则可在时间 $(k/\epsilon)^{O(\log(1/\epsilon)/\epsilon)} \cdot O'(n \cdot h^2 \cdot \log^2(W)) + (k/\epsilon)^{O(1/\epsilon^2)}$ 内得到 $(1+\epsilon, 1+\epsilon)$-近似。若 G 是一棵树，则可在时间 $(k/\epsilon)^{O(\log(1/\epsilon)/\epsilon)} \cdot O'(n^4 \cdot \log^2(W)) + (k/\epsilon)^{O(1/\epsilon^2)}$ 内得到 $(1, 1+\epsilon)$-近似。

此外，我们还可以将结果扩展到图G 经历边插入与删除的动态设置中。本节的第二个主要结果总结如下定理所示。

▶ **定理 3.** 设 $\epsilon > 0$ 和 $k \in \mathbb{N}$。设 $G = (V, E,$ 容量) 为一个具有 n 个顶点且经历边插入与删除的无向加权图。那么，对于 k-平衡划分问题，我们可以维护：

- 若 G 为无权图，则可维护一个 $(n^{o(1)}, 1+\epsilon)$-近似解，其摊销更新时间为 $(k/\epsilon)^{O(\log(1/\epsilon)/\epsilon)} \cdot n^{o(1)} \cdot O'(\log^2(W))$，查询时间为 $(k/\epsilon)^{O(1/\epsilon^2)}$。若 G 是一棵高度为 h的树，则可维护一个 $(1+\epsilon, 1+\epsilon)$-近似解，其最坏情况更新时间为 $(k/\epsilon)^{O(\log(1/\epsilon)/\epsilon)} \cdot O'(h^3 \cdot \log^2(W))$，查询时间为 $(k/\epsilon)^{O(1/\epsilon^2)}$。

我们的动态规划方法受到 Feldmann 和 Foschini [34] 的动态规划启发。然而，Feldmann 和 Foschini 算法中的动态规划单元并非单调，因此无法直接通过单调函数快速卷积方法来加速他们的动态规划。因此，我们首先简化并推广他们的动态规划，使其具有单调性，从而能够应用单调函数快速卷积方法。

我们注意到，在静态和动态算法中，我们可以像在背包问题的命题 12 之后所描述的那样，输出相应的解。

为了得到这些结果，我们首先将在 D.1 节中介绍二叉树这一特殊情形下的精确动态规划。然后，我们将在 D.2 节中通过引入近似，说明如何更高效地计算动态规划。在 D.3 节中，我们将展示如何基于动态规划表返回一个解。D.4 节和 D.5 节分别将讨论如何把二叉树扩展到更一般的图，以及如何扩展到动态设置。

D.1 ExactDP

描述动态规划时，我们将作出两个假设。首先，假设输入图 $T = (V, E)$ 是一棵二叉树（我们将在 D.4 节说明如何删除这一假设）。其次，我们考虑树上的 k-平衡划分问题的一个略微推广版本；需要指出的是，我们在第 4 节没有提到这一推广。在这个推广版本中，我们假设每个顶点都由权重函数 $w\colon V \to \{0, 1\}$.[10]赋予一个权重。为方便起见，对所有 $U \subseteq V$，我们令 $w(U) = \sum_{u \in U} w(u)$，并将 $w(U)$ 称为 U 中顶点的权重。现在，我们的目标是找到 V 的一个划分 $V_1, \ldots, V_k$，使得对所有 i，$w(V_i) \leq (1+\epsilon)\lceil w(V)/k \rceil$；我们将与 OPT $= (V_1^*, \ldots, V_k^*)$ 进行比较，其中 OPT 是满足对所有 i 有 $w(V_i^*) \leq \lceil w(V)/k \rceil$ 的最优解。注意，通过令 $w(v) = 1$

[9] 我们使用记号 $O'(\cdot)$ 来省略 poly(log n，k、log$(1/\epsilon)$、log log(W)) 中的因子。[10] 我们注意到，我们的证明和算法同样适用于更一般的权重函数 $w\colon V \to \mathbb{R}_+$。不过，在这种情况下，我们稍后将引入的函数 DP(v，g、割、$\cdot$) 的计算会变得更加复杂，因此我们在 $\{0, 1\}$ 中采用更简单的顶点权重情形。

对所有 $v \in V$，我们得到标准的 k-平衡划分问题，因此，我们的变体是一个严格推广。

考虑上述推广的原因是，之后我们希望使用我们的算法，借助 Räcke 树 $T = (V, E)$（参见 C.1节）为一般图 $G = (V', E')$寻找平衡划分。然而， G 的顶点 V' 只是 Räcke 树 T 的顶点 V 的一个子集（因为 G 的顶点对应于 T 中的叶节点，而 T 的内部节点不对应于 G中的任何顶点）。因此，如果我们将权重$w(v) = 1$ 赋给 T 中的所有顶点，并计算 T 的一个平衡划分，那么这不一定对应于 G的一个平衡划分。相反，之后我们将考虑这样的权重函数：它为 T 中的所有叶节点（对应于 G中的顶点）赋予权重 1，为 T 的所有内部节点赋予权重 0（在推导G的划分时可以忽略这些节点）。这样， T中的每个集合 V_i 都将对应于 G 中的一个集合 V_i'，满足 $w(V_i) = |V_i'|$。特别是，如果 $w(V_i) \leq (1+\epsilon)\lceil w(V)/k \rceil$，那么我们将得到 $|V_i'| \leq (1+\epsilon)\lceil |V'|/k \rceil$，因此，集合 V_1、……、V_k 给出了 G的一个平衡划分 V_1'、……、V_k'。

动态规划的高层次描述。我们首先对动态规划进行高层次描述。我们从树 G 的叶节点开始，自底向上计算动态规划。对于每个顶点 v，我们都会计算一个代价最小的动态规划解，其中记录了连接到父节点 p 的边是否被割断，以及子树 T_v内部哪些边将被割断；该子树以 v为根节点。请注意，在我们的解中删除割边后，树会被划分为若干互不相交的连通分量，其中恰有一个包含 v的父节点 p。此外，我们还会记录仍与父节点 p （因此也与 T_v外部）相连的顶点的权重信息。我们假设，在计算顶点 v的动态规划单元时，可以使用其两个子节点对应的解。

更具体地说，当我们已经计算出子树 T_v的一个解时，也就是说，我们知道将要删除该子树中节点关联的哪些边（请注意，连接到 v 的父节点的边与 T_v 关联，因此我们将其视为该解的一部分），我们会在动态规划表中存储以下信息。首先，我们存储其代价，即与 T_v 中的顶点关联且被割断的所有边的容量总和。如上所述，我们还希望存储与 v 的父节点相连的顶点的权重，以及 T_v内部连通分量的大小。不过，这里有两个难点：（1）我们无法精确存储与根节点相连的顶点的权重，因为这会导致动态规划表过大。因此，我们存储这样一种代价最低的解：其中权重总和不超过某个给定值的顶点仍与 v的父节点相连。正如我们将看到的，这种方法会产生单调递减函数，并使动态规划表能够得到非常高效的计算。（2）我们会隐式存储删除割边后形成且完全位于 T_v内部的所有连通分量的大小。与前面一样，精确存储这些大小会导致动态规划表过大，因此我们使用签名这一概念对其进行简洁表示。这些签名将帮助我们非常高效地刻画 T_v 内部各连通分量的大小。

签名。若一个连通分量包含的顶点总权重至少为 $\epsilon\lceil w(V)/k \rceil$，则称其为 T_v 大；否则称其为小。令 $t = \lceil \log_{1+\epsilon}(1/\epsilon) \rceil + 1$，并令$M = \lceil k/\epsilon \rceil + 1$。签名是一个向量 $g = (g_0, \ldots, g_{t-1}) \in [M-1]^t$。注意，每个 g_i都是介于 0 和 $M-1$ 之间的整数，因此共有 $M^t = (k/\epsilon)^{O(\epsilon^{-1}\log(1/\epsilon))}$种不同的签名。直观而言， g 中的一项 g_i 大致表示我们所考虑的动态规划解中，权重为$(1+\epsilon)^i \cdot \epsilon\lceil w(V)/k \rceil$的连通分量数量。精确定义如下。

令$\mathcal{S} = \{S_1, \ldots, S_r\}$ 为 T_v 内的一个连通分量集合（例如，可以将$\mathcal{S}$视为

在顶点 v 的动态规划解中移除割边后所产生的连通分量）。若能按以下方式将 S 中的连通分量与 g 中的项匹配，则称签名向量 $g=(g_0,\dots,g_{t-1}) \in [M-1]^t$ 与 S 一致。对于每个大连通分量S_j，令 $\ell(S_j) = \arg\min\{i \in [t]\colon w(S_j) \le (1+\epsilon)^i \cdot \epsilon \mathrm{d} w(V)/k \mathrm{e}\}$，也就是说，$\ell(S_j)$ 是满足 S_j 的权重至多为 $(1+\epsilon)^i \cdot \epsilon \mathrm{d} w(V)/k \mathrm{e}$ 的最小数。令 $s_i \in [M-1]$ 表示在此过程中取值为 $i \in [t]$ 的次数，即 $s_i = |\{j\colon \ell(S_j) = i\}|$，并令 $s=(s_0,\dots,s_{t-1})$ 表示所得向量。若 g 与连通分量集合 S 一致，则称其与 S 一致，即 $g=s$。因此，上述匹配过程可视为对分量大小向上取整，并统计每种大小的连通分量数量。

对于 $x \in \mathbb{N}$，我们令 $e(x) \in [M-1]^t$ 表示总权重为 x 的单个连通分量的签名。更准确地说，我们将 $e(x)$ 设为如下向量：对于使 $e(x)_j = 1$ 成立的 $j = \arg\min\{j \in \mathbb{N}\colon x \le (1+\epsilon)^j \cdot \epsilon \lceil w(V)/k \rceil\}$，其值为 $e(x)_j = 0$；否则为相应的值。如果 $x < \epsilon \lceil w(V)/k \rceil$，我们定义 $e(x) = \vec{0}$。

D.1.1 动态规划定义

现在我们对动态规划进行形式化描述。动态规划表中针对顶点v 的一个条目 DP(v, g, cut x) $\in W_\infty$ 由一个签名 g、一个布尔值 cut 和 $x \in [n]$作为索引。我们将元组 (v, g, cut) 视为动态规划表的行 I，并将 x 视为列；我们为每一行关联函数 DP(v, g, cut, $\cdot$) : $[n] \to W_\infty$。注意，我们的动态规划有 $|V| \cdot M^t \cdot 2 = (k/\epsilon)^{O(\epsilon^{-1}\log(1/\epsilon))} \cdot n$行。此外，它有 n列；之后，尽管 x 只取离散值，我们仍将允许 x 取 $[0,\infty$中的值。

它描述切断与子树 T_v 相连的边所需的最优代价（其中可能包括切断与 v 的父节点相连的边的代价）。我们将删除割边后，T_v 中仍与 v 的父节点相连的顶点集合称为<款 id='5'>根连通分量</款>。我们对 DP(v, g,割 x)施加以下条件：

- 删除割边后，根连通分量 $U \subseteq T_v$ 的总权重至多为 x，即 $w(U) \le x$。如果将割设为 true，则切断 v 与其父节点之间的边，否则保留该边。删除割边后，T_v 中与 v 的父节点不相连的顶点所形成的连通分量与签名 g 一致。

我们注意到，如果固定顶点 v、签名 g 和割的取值，那么所得函数 DP(v, g, 割,$\cdot$) 关于 x 单调递减。这是本节其余部分的关键性质。

▶ **观察 23.** 设 $v \in V$, $g \in [M-1]^t$ 是一个签名，*cut* 为 $\in$ *{真,假}*。那么函数 动态规划 (v, g, *cut*, $\cdot$) : $[0,\infty) \to \mathbb{R}_+$ 单调递减。

证明。根据定义，DP(v, g, cut x)存储的是根连通分量中的顶点权重至多为 x时的最优解代价。现在注意，对于 $x \le x'$，解 DP(v, g, cut x)也是 DP(v, g, cut x')的一个可行解。因此，DP(v, g, cut, $\cdot$)必定单调递减。◀

由于动态规划单元关于 x单调递减，我们将使用简写记法 DP(v, g, 割 ∞) 来表示解 $\min_x$ DP(v, g, 割 x)。注意，这个最小值是在 DP(v, g, 割, $\cdot$)发生变化时对应的最大 x值处取得的。

D.1.2 计算动态规划

接下来，我们将说明如何精确计算 DP(v,$\cdot$,$\cdot$,$\cdot$)。要计算 DP(v,$\cdot$,$\cdot$,$\cdot$)，只需遍历 x、g 和 割 的所有可能取值。注意，由于每个顶点的

权重属于$\{0,1\}$，函数 DP$(v,g,$割$,\cdot)$ 仅在 $x\in[n+1]$ 时发生变化（即当 x 为整数时）。因此，我们只需考虑 $n+1$ 个 x 取值。由此可知，对于固定顶点 v，计算 DP$(v,\cdot,\cdot,\cdot)$ 时，需要遍历 $O(M^t\cdot n)$ 种参数取值。

在我们的描述中，我们用 p 表示 v的父节点，并分别用 v_l 和 v_r 表示 v的左子节点和右子节点（如果存在）。

情况 1： v **是叶节点**。如果我们割断与 v的父节点相连的边，那么代价为 cap$(v,\ p)$；根连通分量中没有顶点，而 v 形成一个独立的连通分量，其签名为 $e(w(v))$。因此，对于所有 $x\in[0,\infty)$，我们设 DP$(v,\ e(w(v)),$ true $x)=$cap$(v,\ p)$；并且对于所有 $x\in[0,\infty)$ 以及所有签名 $g\neq e(w(v))$，我们设 DP$(v,\ g,$ true $x)=\infty$。

现在假设我们不割断与 v的父节点相连的边 $(v,\ p)$。由于没有割断任何边，因此无需支付任何代价；根连通分量中的顶点权重为 $w(v)$，签名为 $g=0$，因为 T_v 中不存在未与 p相连的连通分量。因此，对于所有 $x\in[0,w(v))$，我们设 DP$(v,\ 0,$ false $x)=\infty$；对于所有$x\in[w(v)\ \infty)$，我们设 DP$(v,\ 0,$ false $x)=0$。对于所有签名 $g\neq 0$ 以及所有 $x\in[0,\infty)$，我们设 DP$(v,\ g,$ false $x)=\infty$。

情况 2： v **不是叶节点**。如果 v 不是叶节点，则假设它恰好有两个子节点v_l 和 v_r（如果它只有一个子节点，我们可以添加第二个子节点 v'，使其具有 $w(v')=0$、容量 cap$(v,v')=0$，这样 v' 就不会影响解）。假设对于 v_l 和 v_r，我们已经针对 x、g 和 割的所有可能取值，计算出了动态规划$(v_l,g,$割 $x)$和动态规划$(v_r,g,$割 $x)$的解。

令 $e_l=(v,v_l)$ 和 $e_r=(v,v_r)$ 表示分别通向子节点的边，并令 $e_p=(p,v)$ 表示通向 v的父节点 p 的边。接下来，我们根据决定割掉这些边中的哪些边，区分四种情况（A、B、C、D）。对于每种情况，我们计算动态规划$_{\text{case}}(v,g,$割 $x)$值，即在按照该情况割掉 e_l 和 e_r 的条件下所得到的最优值。最终的条目动态规划$(v,g,$割 $x)$则通过对所有情况取最小值获得，即令

$$\mathsf{DP}(v,g,cut,x)=\min_{\text{case}\in\{A,B,C,D\}}\mathsf{DP}_{\text{case}}(v,g,cut,x)$$

对于所有 x、g，。

情况 A：割 e_l **和** e_r。假设我们割掉 e_l 和 e_r。那么，给定 x 和 g，我们必须为左子树和右子树选择子解，使得能够到达 p的顶点权重至多为 x，并且其中的连通分量与 g一致。

首先，假设我们割掉边 e_p。那么，割掉这条边的代价为 cap(v,p)。此外，T_v 内部能够到达 p 的顶点权重为零，因此，根据 DP$(v,g,$ cut$,\cdot)$ 的单调性，x 的值无关紧要。接下来，如果左子树和右子树中分别存在签名为 g_l 和 g_r 的解，我们可以将这些解组合起来，只要 $g_l+g_r+e(w(v))=g$（因为我们割掉了边 e_l 和 e_r，顶点 v 形成了一个权重为 $w(v)$ 的单个连通分量）。注意，在子节点 v_l 的子解中，x 的值不会影响解 DP$(v,g,$ cut $x)$ 的可行性，因为 T_{v_l} 中的根连通分量大小已经编码在 g_l中。因此，为了获得最小值代价，我们考虑 DP$(v_l,g_l,$ true $\infty)$；根据对称性，对于 v_r也同样成立。因此，我们对所有$x\in[0,\infty)$进行设置，

$$\text{动态规划}_A(v,g,\text{真}\ x)=\text{容量}(v,p)+\min_{g_l+g_r=g-e(w(v))}\{\text{动态规划}(v_l,g_l,\text{真}\ \infty)+\text{动态规划}(v_r,g_r,\text{真}\ \infty)\}\quad(5)$$

其次，假设我们不割与父节点 p 相连的边。那么至少有一个顶点（即 v）可以到达 p。因此，对于所有签名 g，DP$_A(v,g,$ false $x)=\infty$ for

$x \in [0, w(v))$. 对于 $x \in [w(v)\ \infty)$，我们可以像上面那样组合这些解，并设定

$$\mathsf{DP}_A(v, g, \text{false}, x) = \min_{g_l+g_r=g} \{\mathsf{DP}(v_l, g_l, \text{true}, \infty) + \mathsf{DP}(v_r, g_r, \text{true}, \infty)\}. \tag{6}$$

情况 B：既不切割 e_l 也不切割 e_r。 接下来，假设我们既不切割 e_l，也不切割 e_r。在这种情况下，我们必须为 T_{v_l} 和 T_{v_r}选择子解，其中每个子解由上界 x_l（分别为 x_r）及其签名 g_l（分别为 g_r）确定。

首先，假设我们切割边 e_p。如果令 x_l 和 x_r 表示子解的根连通分量的精确权重，那么顶点 v 之后将被包含在大小为 $x_l + x_r + w$（v）的连通分量中。因此，只要 $g_l + g_r + e$（$x_l + x_r + w$（v）） $= g$，我们就可以将这些子解合并为签名为 g 的解。因此，我们对每个$x \in [0, \infty)$都设定，

$$\mathsf{DP}_B(v, g, \text{true}, x) = \\ \text{cap}(v, p) + \min_{x_l, x_r, g_l+g_r=g-e(x_l+x_r+w(v))} \mathsf{DP}(v_l, g_l, \text{false}, x_l) + \mathsf{DP}(v_r, g_r, \text{false}, x_r).$$

其次，假设我们不切割 e_p。那么我们同样必须对所有签名 g 和所有 $x \in [0, w$（v）），将 DP_B（v，g，false x）$= \infty$设为某值，因为权重为 w（v）的顶点 v 可以到达 p。对于$x \geq w$（v），我们必须选择 x_l和 x_r，使它们的和为 $x - w$（v），从而确保权重至多为 x 的顶点可以到达父节点 p。因此，我们对所有 $x \in [w$（v） $\infty)$都设定

$$\mathsf{DP}_B(v, g, \text{false}, x) = \min_{g_l+g_r=g, x_l+x_r=x-w(v)} \mathsf{DP}(v_l, g_l, \text{false}, x_l) + \mathsf{DP}(v_r, g_r, \text{false}, x_r).$$

情况 C：割 e_l 但不割 e_r。 现在假设我们割断通向左子节点 v_l 的边，但不割断通向右子节点 v_r的边。在这种情况下， v 仍与v_r 的根连通分量连通，因此我们需要为 T_{v_r} 选择一个参数为 x_r 和 g_r 的子解，并为 T_{v_l}选择一个参数为 g_l 的子解。注意，由于我们割断了 e_l，v_l 的根连通分量权重上界无关紧要，因为它已隐式编码在 g_l中。

首先，假设我们割断 e_p。如果令 x_r 表示 T_{v_r} 中子解的根连通分量的精确权重，那么之后 v 将属于一个大小为 $x_r + w(v)$ 的连通分量。因此，只要 $g_l + g_r + e(x_r + w(v)) = g$，我们就可以将这些子解合并为签名为 g的一个解。因此，对于每个 $x \in [0, \infty)$，我们设置

$$\mathsf{DP}_C(v, g, \text{true}, x) = \text{cap}(v, p) + \min_{x_r, g_l+g_r=g-e(x_r+w(v))} \mathsf{DP}(v_l, g_l, \text{true}, \infty) + \mathsf{DP}(v_r, g_r, \text{false}, x_r). \tag{7}$$

其次，假设我们不割断 e_p。那么对于所有签名 g 以及所有 $x \in [0, w(v))$，我们都必须设置动态规划$_C(v,\ g,$ false $x) = \infty$，因为权重为 $w(v)$ 的顶点 v 可以到达 p。对于$x \in [w(v)\ \infty)$，我们必须选择 $x_r \leq x - w(v)$，这样可以保证总权重至多为 x 的顶点能够到达父节点 p。根据动态规划$(v_r,\ g_r,$ false，$\cdot)$ 的单调性，我们只需选择 $x_r = x - w(v)$。因此，对于所有 $x \in [w(v)\ \infty)$，我们设置

$$\mathsf{DP}_C(v, g, \text{false}, x) = \min_{g_l+g_r=g, x_r=x-w(v)} \mathsf{DP}(v_l, g_l, \text{true}, \infty) + \mathsf{DP}(v_r, g_r, \text{false}, x_r). \tag{8}$$

情况 D：割 e_r 但不割 e_l。 与情况 C 对称。

接下来，我们论证这个动态规划是良态的，即满足定义 17。特别地，我们注意到，这个动态规划并非良好态，因为它不满足

定义 8 的性质 (4b)：在下文的情况 2、步骤 B 中，我们将不得不执行过多的最小值操作（见公式（11））。我们还将证明，这个动态规划的依赖图恰好就是输入树，因此满足引理 19 的条件。此外，只需枚举上述各个最小值操作中的所有选项，即可在 $O(M^{2t}n^3)$ 时间内计算动态规划单元 DP$(v, \cdot, \cdot, \cdot)$ 的所有条目。

▶**引理 24.** 该动态规划是良态的，且依赖树与输入树 T 完全相同。此外，给定一个顶点 v，我们可以在时间$O(M^{2t}n^3)$内计算 动态规划$(v,\cdot,\cdot,\cdot)$ 中的所有表项。

证明。 首先，注意在动态规划中，每个单元 动态规划$(v,\cdot,\cdot,\cdot)$ 只依赖于其两个子节点的解。注意，这些恰好就是依赖图中存在且也在 T中的边。因此，依赖图与 T 完全相同。此外，当子节点解的输入是一个 β-近似时，由于所有计算都是精确进行的，动态规划的输出也将是一个 β-近似。因此，该动态规划也是良态的。

其次，我们来考虑运行时间。回顾一下，对于固定的 x, g 和 割，我们设 DP$(v, g,$割 $x) = \min_{\text{case}\in\{A,B,C,D\}}$ DP$_{\text{case}}(v, g,$ 割 $x)$，这个量可以通过简单的表查找在时间 $O(1)$ 内计算得到。因此，我们只需考虑在每种情况 $\in \{A, B, C, D\}$ 下，以及固定 x, g和 割时，计算 DP$_{\text{case}}(v, g,$ 割 $x)$ 所需的时间。

对于情况 A，注意到可以通过遍历所有 M^t 个 g_l 的取值，并在 g_r 为非负向量时设置 $g_r = g - e(w(v)) - g_l$，来计算最小值操作。然后，可以通过常数时间的表查找计算最小值项中的表达式。因此，时间复杂度为 $O(M^t)$。对于情况 B，若割 $=$ 为真，注意我们可以遍历 x_l、 x_r 的所有取值，并按上述方法遍历 g_l 。这需要 $O(M^t n^2)$ 的时间。若割 $=$ 为假，我们可以再次按上述方法遍历 g_l，并遍历所有 $x_l \in [n+1]$，在 $x_r \geq 0$时设置 $x_r = x - w(v) - x_l$；因此，该情况可在 $O(M^t n)$ 的时间内求解。对于情况 C 和 D，我们可以遍历 x_r 的所有取值，然后按上述方法遍历 g_l 。总运行时间为 $O(M^t n)$。

我们得出，对于固定的 x、 g 和 割，计算所有情形 $\in \{A, B, C, D\}$ 下 DP$(, g,$割 $x)$ 所需的时间为 $O(M^t n^2)$。由于 x 有 $O(n)$ 种选择， M^t 有 g种选择，而 割 有两种选择，我们得出，计算 DP$(v,\cdot,\cdot,\cdot)$ 的总运行时间为$O(M^{2t}n^3)$。◀

D.2 近似动态规划

本节将说明如何高效地构造近似动态规划表。为此，我们基本上执行与上述相同的计算，但不再通过计算各情形 DP$_{\text{case}}(v,\cdot,\cdot,\cdot)$ 的精确解来计算精确解 DP$(v,\cdot,\cdot,\cdot)$，而是计算近似解 ADP$(v,\cdot,\cdot,\cdot)$，它是近似解 ADP$_{\text{case}}(v,\cdot,\cdot,\cdot)$ 中的最小值，其中情形为 $\in \{A, B, C, D\}$。

不过，这里有几个关键区别。首先，对于固定的 v、 g、割 和情形 $\in \{A, B, C, D\}$，我们将 ADP$_{\text{case}}(v, g,$ 割，$\cdot)$ 解释为一个分段常数函数，并以高效的列表表示形式存储（见第 2 节）。计算出解 ADP$_{\text{case}}(v, g,$ 割，$\cdot)$ 后，我们计算函数

$$\begin{aligned}&\mathsf{ADP}(v, g, cut, \cdot) :=\\ &\lceil \min\{\mathsf{ADP}_A(v, g, cut, \cdot), \mathsf{ADP}_B(v, g, cut, \cdot), \mathsf{ADP}_C(v, g, cut, \cdot), \mathsf{ADP}_D(v, g, cut, \cdot), \}\rceil_{1+\delta},\end{aligned} \tag{9}$$

也就是说，我们不只是对不同情形取最小值，还会将结果舍入到 $1+\delta$ 的倍数。根据引理 6，这个舍入步骤会引入 $\alpha = 1+\delta$ 的近似误差，但会将分段常数函数 DP(v， g， 割，·) 中分段的数量减少至$p := O(\log_{1+\delta}(W))$（要使这一方法奏效，我们必须保证待舍入的函数是单调的，因此我们将证明，对于每种情形 $\in \{A, B, C, D\}$，ADP$_{\text{case}}$(v， g， 割，·) 都是单调的）。第二个关键区别当然是，我们执行上述计算时使用的值已经经过舍入；也就是说，使用的是 ADP 中的条目，而非 DP 中的条目。我们注意到，公式（9）是近似动态规划中唯一 不精确的地方；其他所有计算都精确进行（不进行任何舍入），因此近似动态规划只损失一个 $1+\delta$ 因子。

为了保证实现具有极高的效率，我们依赖近似动态规划中各项的以下不变量：

1. 对所有 v、 g 和割，函数 ADP($v, g,$ cut, ·) 均单调递减。2. 对所有 v、 g 和割，函数 ADP($v, g,$ cut, ·) 为分段常数，且至多有$p := O(\log_{1+\delta}(W))$ 个分段。

请注意，第一个性质类似于这样一个事实：对于精确动态规划，根据观察 23，DP(v， g、割，·) 单调递减。然而，这里我们将该性质表述为一个不变量，因为 DP(v， g、割，·) 可能存在非单调的近似，因此我们需要证明我们的每个函数 ADP(v， g、割，·) 确实是单调的。请注意，第二个性质可由公式（9）中的单调性和舍入步骤直接推出，因此下文无需证明这一点。

与精确动态规划的描述类似，我们现在将逐一讨论各种情形，并且，˜给定 v，说明如何在时间 $O(1)$ 内对所有 g和 割 计算 ADP(v， g， 割，·)。这些情形与精确动态规划中的完全相同，因此为简洁起见，我们不再重复正确性论证。

情形 1： v 是一个叶节点。然后，我们按照精确情形中的做法处理。对于所有 $x \in [0, \infty)$，我们设置 ADP($v, e(w(v))$, true x) =容量(v, p)；对于所有 $x \in [0, \infty)$ 以及所有签名 $g \neq e(w(v))$，我们设置 ADP($v, g,$ true x) $= \infty$。此外，对于所有 $x \in [0, w(v))$，我们设置 ADP(v, 0, false x) $= \infty$；对于所有 $x \in [w(v)\ \infty)$，我们设置 ADP(v, 0, false x) $= 0$。对于所有签名 $g \neq 0$ 以及所有$x \in [0, w(v))$，我们设置 ADP($v, g,$ false x) $= \infty$。请注意，在所有情形下，相应的函数 ADP($v, g,$ 割, ·) 都是单调递减的，并且有 $O(1)$ 个分段。

情况 2： v 不是叶节点。我们区分与精确动态规划相同的四种情况。同样，我们假设 v 恰有两个子节点 v_l 和 v_r，并令 $e_l = (v, v_l)$、 $e_r = (v, v_r)$ 和 $e_p = (p, v)$，其中 p 是 v的父节点。

情况 A：割 e_l 和 e_r。 首先，假设我们割掉 e_l 和 e_r。然后，与精确动态规划中一样，如果我们割掉通向 v的父节点的边，我们希望设定

$$\begin{aligned}&\mathsf{ADP}_A(v, g, \text{true}, x) = \\ &\text{cap}(v, p) + \min_{g_l + g_r = g - e(w(v))} \{\mathsf{ADP}(v_l, g_l, \text{true}, \infty) + \mathsf{ADP}(v_r, g_r, \text{true}, \infty)\}.\end{aligned}$$

对所有 $x \in [0, \infty)$。注意，在上面的等式中，量 cap(v, p)、ADP($v_l, g_l,$ true ∞) 和 ADP(v_r, g_r,true ∞) 都只是数值，因此可以视为只有一个分段的分段常数函数。因此，ADP$_A$($v, g,$ true,·) 是只有一个分段的分段常数函数，因此也是单调递减的。于是，ADP$_A$($v, g,$ true,·) 满足这些不变量。此外，ADP$_A$($v, g,$ true, ·) 可通过引理 6，利用对单调递减的分段函数求和与取最小值来计算。注意，取最小值会得到 $O(M^t)$ 个不同的值，因为计算时会遍历所有

$g_l \in [M-1]^t$ 并设置 $g_r = g - e(w(v)) - g_l$，前提是 g_r 中的所有项均为非负数。根据我们的不变量，每个函数 $\mathrm{ADP}(v_l, g_l, \mathrm{true}, \cdot)$ 都有 $O(p)$ 个分段，因此我们可以在 $O(1)$ 时间内计算 $\mathrm{ADP}(v_l, g_l, \mathrm{true}\ \infty)$ 的值；$\mathrm{ADP}(v_r, g_l, \mathrm{true}\ \infty)$ 也同样如此。因此，计算 $\mathrm{ADP}_A(v, g, \mathrm{true}, \cdot)$ 需要 $O(M^t)$ 时间。

接下来，假设我们不割断与 v 的父节点相连的边。那么，与精确动态规划中一样，我们希望设置：

$$\mathrm{ADP}_A(v, g, \mathrm{false}, x) = \min_{g_l + g_r = g} \{\mathrm{ADP}(v_l, g_l, \mathrm{true}, \infty) + \mathrm{ADP}(v_r, g_r, \mathrm{true}, \infty)\}$$

对所有 $x \in [0, \infty)$。那么，根据与上文相同的论证，$\mathrm{ADP}_A(v,\ g,\ \mathrm{false},\ \cdot)$ 是一个仅含一个分段的分段常值单调递减函数。按照上文所述，可在时间 $O(M^t)$ 内计算得到。

情况 B：既不割断 e_l，也不割断 e_r。 现在假设我们不割断与子节点相连的任何边。

如果我们不割断与 v 的父节点相连的边，就按照与精确动态规划类似的方式进行。首先设置 $\mathrm{ADP}_B(v,\ g,\ \mathrm{false}\ x) = \infty$，对所有 $x \in [0, w(v))$。接下来，对于 $x \in [w(v)\ \infty)$，我们希望设置

$$\begin{aligned}\mathrm{ADP}_B(v, g, \mathrm{false}, x) &= \min_{g_l + g_r = g, x_l + x_r = x - w(v)} \mathrm{ADP}(v_l, g_l, \mathrm{false}, x_l) + \mathrm{ADP}(v_r, g_r, \mathrm{false}, x_r) \\ &= \min_{g_l + g_r = g} \min_{x_l + x_r = x - w(v)} \mathrm{ADP}(v_l, g_l, \mathrm{false}, x_l) + \mathrm{ADP}(v_r, g_r, \mathrm{false}, x_r).\end{aligned} \tag{10}$$

注意，对于固定的 g_l 和 g_r，第二行中的内层最小值运算描述了一个（min $+$）-卷积，这是由约束条件 $x_l + x_r = x - w(v)$ 导致的。因此，在内层最小值运算中，我们计算卷积 $\mathrm{ADP}(v_l, g_l, \mathrm{false}, \cdot) \oplus \mathrm{ADP}(v_r, g_r, \mathrm{false}, \cdot)$，并通过引理 6 中的平移操作将结果平移 $w(v)$（其中，对于 $x \in [0, w(v))$，我们设置 $\mathrm{ADP}_B(v, g, \mathrm{false}\ x) = \infty$）。根据引理 7，计算该卷积需要时间 $O(p^2 \log p)$。为了计算方程（10）中的外层最小值，我们遍历所有 $g_l \in [M-1]^t$，因此需要对至多含有 p^2 个分段的分段常数函数执行 $O(M^t)$ 次最小值计算。根据引理 21，这需要时间 $O(M^t p^2 \log(M^t p^2))$。根据引理 22，$\mathrm{ADP}_B(v, g, \mathrm{false}, \cdot)$ 是单调递减的，因为它是两个单调递减函数的卷积的最小值。

如果我们切断连接到 v 的父节点的边，那么对于所有 $x \in [0, \infty)$，我们希望设置

$$\mathrm{ADP}_B(v,\ g,\ \mathrm{true}\ x) =$$
$$\text{容量}(v, p) + \min_{x_l,\ x_r,\ g_l + g_r = g - e(x_l + x_r + w(v))} \mathrm{ADP}(v_l,\ g_l,\ \mathrm{false}\ x_l) + \mathrm{ADP}(v_r,\ g_r,\ \mathrm{false}\ x_r)。$$

请注意，这里需要谨慎，因为 g_l 和 g_r 的值域取决于 $x_l + x_r$ 的选择。由于 $x_l + x_r$ 有 $\Omega(n)$ 个可能的取值，我们无法遍历 $x_l + x_r$ 可以取到的所有值。相反，我们将利用 $\mathrm{ADP}(v_l,\ g_l,\ \mathrm{false},\ \cdot)$ 和 $\mathrm{ADP}(v_r,\ g_r,\ \mathrm{false},\ \cdot)$ 的单调性，证明只需考虑 $O(\log(k/\epsilon)/\epsilon)$ 个不同的数对 $(x_l,\ x_r)$。

首先，注意我们可以假设 $x_l \le w(T_{v_l})$ 和 $x_r \le w(T_{v_r})$：进一步提高根连通分量权重的上界，意味着根连通分量所含权重超过子树中所有顶点的权重总和，这是不可能的。因此，$x_l + x_r + w(v) \in [1, w(V)]$。

其次，我们将区间 $[1, w(V)]$ 划分为 $O(\log(k/\epsilon)/\epsilon)$ 个区间。我们有区间 $I_j = (\xi_{j-1}, \xi_j]$，其中 $\xi_j = \lceil (1+\epsilon)^j \epsilon w(V)/k \rceil$ 对所有 $j = 1, \ldots, \log_{1+\epsilon}(k/\epsilon)$ 都成立。此外，我们添加一个“区间” $I_0 := [\lceil \epsilon w(V)/k \rceil\ \lceil \epsilon w(V)/k \rceil]$，以及区间 $I_{-1} := [1, \lceil \epsilon w(V)/k \rceil)$。我们设定 $\xi_0 = \lceil \epsilon w(V)/k \rceil$，并将 ξ_{-1} 设为小于 $\lceil \epsilon w(V)/k \rceil$ 的最大整数。注意

对于所有 $j \geq -1$ 和 $x \in I_j$，我们都有 $e(x) = e(\xi_j)$，也就是说，$e(x)$ 的值在区间 I_j 上保持不变。下文中，这一性质将使我们能够分别处理关于 $x_l + x_r$ 和关于 $g_l + g_r$ 的条件。

现在，我们可以将上述表达式改写为

$\mathrm{ADP}_B(v, g, \text{true } x) = \text{容量}(v, p) + \text{最小值}_j \text{最小值}_{x_l+x_r+w(v)\in I_j} \text{最小值}_{g_l+g_r=g-e(\xi_j)}$
$\mathrm{ADP}(v_l, g_l, \text{false } x_l) + \mathrm{ADP}(v_r, g_r, \text{false } x_r)$。

第三，请注意，现在这两个最小值操作只取决于 j 的选择，并且重要的是，对 g_l 和 g_r 取最小值不再取决于 $x_l + x_r$ 的选择。因此，我们可以交换这两个最小值操作的顺序。此外，由于 $\mathrm{ADP}_B(v, g, \text{false } x)$ 关于 x单调递减，我们可以将 x_l和x_r 的选择限制为使得 $x_l + x_r + w(v)$ 是相应区间 I_j 中最大的数，即$x_l + x_r + w(v) = \xi_j$。因此，

$$\mathrm{ADP}_B(v, g, \text{true } x) = \text{容量}(v, p) + \text{最小值}_j \text{最小值}_{g_l+g_r=g-e(\xi_j)} \text{最小值}_{x_l+x_r+w(v)=\xi_j} \mathrm{ADP}(v_l, g_l, \text{false } x_l) + \mathrm{ADP}(v_r, g_r, \text{false } \xi_j - x_l - w(v))。 \tag{11}$$

接下来，我们解释如何高效地计算上述表达式。首先，我们说明如何高效地计算上述表达式中的内部最小值运算。我们首先注意到，该最小值运算不是卷积，因为在约束条件中，我们将求和至ξ_i，其中ξ_i是一个常数（而不是变量 x）。现在回顾一下，根据我们的不变量，$\mathrm{ADP}(v_l, g_l, \text{false}, \cdot)$ 和 $\mathrm{ADP}(v_r, g_r, \text{false}, \cdot)$ 是具有 $O(p)$个分段的分段常数函数。由于$x_l, x_r \geq 0$，这意味着对于 x_l和 x_r，只有 $O(p^2)$种选择，使得 $x_l, x_r \in I_j$，并且要么在 $\mathrm{ADP}(v_l, g_l, \text{false } x_l)$ 中开始一个新的分段，要么在 $\mathrm{ADP}(v_r, g_r, \text{false } x_r)$ 中开始一个新的分段。因此，我们可以遍历所有这些对 (x_l, x_r)，并计算 $\mathrm{ADP}(v_l, g_l, \text{false } x_l) + \mathrm{ADP}(v_r, g_r, \text{false } x_r)$，其中 $x_r = \xi_j - x_l - w(v)$。这样，我们可以在 $O(p^2 \log p)$时间内计算内部最小值运算。

接下来，我们只需遍历 j 以及 g_l 的所有取值，并按照上文所述，在 $O(M^t \cdot \log(k/\epsilon)/\epsilon)$ 次迭代中设置 $g_r = g - e(\xi_j) - g_l$，即可计算外层的两个最小值操作。因此，运行时间为 $O(M^t p^2 \log p \cdot \log(k/\epsilon)/\epsilon)$。我们注意到，这一步使其成为良态而非良好态（因为它违反了定义 8 的性质 (4b)）。

最后，我们注意到，由于 $\mathrm{ADP}_B(v, g, \text{true } x)$ 与 x无关，因此它是一个常数。因此，$\mathrm{ADP}_B(v, g, \text{true } x)$ 是一个仅含一个部分的分段常数函数，并且单调递减。

情况 C：割 e_l 但不割 e_r。 现在假设我们割去通向左子节点的边，但不割去通向右子节点的边。

首先假设我们割去通向 v的父节点的边。与精确动态规划中一样，对于所有 $x \in [0, \infty)$，我们希望设置

$$\mathrm{ADP}_C(v, g, \text{true}, x) = \mathrm{cap}(v, p) + \min_{x_r, g_l+g_r=g-e(x_r+w(v))} \mathrm{ADP}(v_l, g_l, \text{true}, \infty) + \mathrm{ADP}(v_r, g_r, \text{false}, x_r).$$

与前一种情况一样，注意到在最小值中，约束 $g_l + g_r = g - e(x_r + w(v))$ 取决于 x_r 的选择。因此，我们仿照前面的做法，将上述方程改写为

情况：

$$\begin{aligned}
&\mathsf{ADP}_C(v,g,\text{true},x)\\
&\quad=\text{cap}(v,p)+\min_j \min_{x_r+w(v)\in I_j}\ \min_{g_l+g_r=g-e(\xi_j)} \mathsf{ADP}(v_l,g_l,\text{true},\infty)+\mathsf{ADP}(v_r,g_r,\text{false},x_r)\\
&\quad=\text{cap}(v,p)+\min_j \min_{g_l+g_r=g-e(\xi_j)}\ \min_{x_r+w(v)\in I_j} \mathsf{ADP}(v_l,g_l,\text{true},\infty)+\mathsf{ADP}(v_r,g_r,\text{false},x_r)\\
&\quad=\text{cap}(v,p)+\min_j \min_{g_l+g_r=g-e(\xi_j)} \mathsf{ADP}(v_l,g_l,\text{true},\infty)+\mathsf{ADP}(v_r,g_r,\text{false},\xi_j-w(v)),
\end{aligned}$$

其中在最后一步中，我们用到了 $\mathsf{ADP}(v_r,\ g_r,\ \text{false},\ \cdot)$ 是单调递减的。计算两个具有 $O(p)$ 个分段的分段常值函数的函数值，所需时间为 $O(\log(p))$。此外，通过穷举 j 的所有选择，并对 g_l 和 g_r 按照上述方法进行处理，我们得到 $O(M^t\log(k/\epsilon)/\epsilon)$ 次迭代，总运行时间为 $O(M^t\log\ p\log(k/\epsilon)/\epsilon)$。与之前一样，$\mathsf{ADP}_C(v,\ g,\ \text{true},\ \cdot)$ 是一个常数（因为计算不依赖于 x），因此它只有一个分段，并且单调递减。

接下来，假设我们不割断通向 v 的父节点的边。于是，对于所有签名 g 和所有 $x\in[0,w(v))$，我们令 $\mathsf{DP}_C(v,\ g,\ \text{false},\ 0)=\infty$。对于所有 $x\in[w(v)\ \infty)$，我们令

$$\mathsf{ADP}_C(v,g,\text{false},x)=\min_{g_l+g_r=g}\mathsf{ADP}(v_l,g_l,\text{true},\infty)+\mathsf{ADP}(v_r,g_r,\text{false},x-w(v)).$$

注意，在最小值运算中，第一项是常数，第二项是经过 $w(v)$位移的分段常数函数。此外，最小值是在 $O(M^t)$个分段常数函数上取的（每个函数对应于 g_l 的一种选择，理由与上文相同）。我们可以通过引理 6 执行加法和平移操作（每次应用的运行时间为 $O(p\log\ p)$）。然后，我们对 M^t 个函数执行最小值运算，其中每个函数只有 p 个分段。根据引理 21，这可以在 $O(M^tp\log(M^tp))$时间内完成。总运行时间为 $O(M^tp\log(M^tp))$。

情况 D：割 e_r，但不割 e_l。与情况 C 对称。

我们用以下引理总结本小节，并概括近似动态规划计算的性质。该引理可由上述讨论立即得出。

▶ **引理 25.** 近似动态规划计算得到一个$(1+\delta)$-近似动态规划解，且依赖树与输入树 T 相同。给定一个顶点 v、一个签名 g 以及取值割 $\in${真,假}，我们可以在 $O(M^tp^2\log(M^tp)\log(k/\epsilon)/\epsilon))$时间内计算相应的近似动态规划表项$\mathsf{ADP}(v,g,$ 割$,\cdot)$。

证明。 近似动态规划的近似比为 $(1+\delta)$近似，因为正如我们之前所指出的，除了公式（9）中的舍入步骤外，我们只使用精确计算。因此，在计算中我们只使用了 $(1+\delta)$因子。

关于运行时间的结论可直接由引理上方的讨论得出，我们已经在那里分析了所有步骤的运行时间。◀

D.3 计算结果

本节介绍如何使用前面描述的动态规划来求解 k-平衡划分问题。回顾一下，我们考虑的是 k-平衡划分问题的广义版本，其中每个顶点 v 都有一个权重 $w(v)\in\{0,1\}$（定义见 D.1 节）。

我们关注该问题的数值版本，在这个版本中，我们只需输出最优割 OPT 的值 的一个近似值，而无需返回取得该割值的实际划分 $V_1, \ldots, V_k$。不过需要指出，通过从上到下分析动态规划解，我们也可以在时间$O(n)$内构造出一个具体划分 $V_1, \ldots, V_k$，使其达到数值版本所返回的割值。

可行签名。在介绍我们的算法之前，我们首先需要引入可行签名的概念。具体而言，回顾一下，在 D.1 节中，我们将签名介绍为一种简洁的方式，用于存储一个解中各连通分量的大小。现在，可行签名指的是这样一种签名：其中的连通分量可以划分，从而得到一个近乎 k 平衡的顶点划分。下面将对这一想法进行更形式化的说明。

对于每个签名 $g = (g_0、\ldots\ldots、g_{t-1}) \in [M-1]^t$，我们称其关联的机器调度实例[11] $I(g)$为恰好包含所有 i 中大小为 $(1+\epsilon)^i \cdot \epsilon \lceil w(V)/k \rceil$ 的 g_i 个作业的实例。若实例 g 中的作业可以调度到k 台机器上，且最大完工时间至多为 $(1+\epsilon)\lceil w(V)/k \rceil$，则称 g 为一个可行签名。稍后，我们将把调度问题中的机器对应到 k 平衡划分解中的划分，并把作业对应到连通分量。这样，我们就能确保 k 平衡划分解满足平衡约束。

算法。现在我们介绍适用于二叉树的两种静态算法。这两种算法的唯一区别在于使用 C.2 节中的精确动态规划，还是使用 D.2 节中的近似动态规划；我们分别将它们称为精确算法和近似算法。假设输入为误差参数 $\epsilon > 0$ 和一棵有根加权树 $T =（V，E，$容量），其根节点为 r，顶点权重为 $w（v）\in \{0，1\}$，我们希望求解 k-平衡划分问题。

首先，我们通过添加一个虚根 r'来扩充 T。令 r' 为 r 的父节点，并设置 $w（r'）= 0$ 和 cap$(r，r') = 0$。然后，我们按照 C.2 节所述自底向上计算动态规划，并将 T 视为其自身的依赖图。在精确算法中，我们使用 D.1 节中的动态规划；在近似算法中，我们使用 D.2 节中的动态规划。

其次，我们计算所有近似可行签名的集合。为得到该集合，我们枚举所有 M^t 签名，并逐一检查它们是否近似可行。具体做法如下。对于每个签名 g，我们构造机器调度实例 $I(g)$，并针对该问题运行由 Hochbaum 和 Shmoys 提出的多项式时间近似方案 [39]，其近似比为 $1+\bar{\epsilon}$，运行时间为$(N/\bar{\epsilon})^{O(1/\bar{\epsilon}^2)}$，其中 N 表示 $I(g)$中的作业总数，稍后我们将看到，如果 k、ϵ 和 $\bar{\epsilon}$ 是常数，那么 N 也是常数。如果 $I(g)$返回的最大完工时间至多为$(1+\bar{\epsilon})(1+\epsilon)\lceil w(V)/k \rceil$，我们就将签名 g 加入近似可行签名集合。需要指出的是，使用多项式时间近似方案时，我们计算出的集合可能包含一些不可行的签名，但它们仍不会过度违反平衡约束。

第三，我们考虑树的（真实）根节点 r 处动态规划表中的表项，前提是连接到其（人工）父节点的边未被割掉（回想一下，我们添加了一条权重为 0 的边，从真实根节点 r 连接到虚根 r'，因此割掉它不会产生任何代价），也就是说，根据我们处于近似情形还是精确情形，我们分别考虑动态规划表项 DP$(r，\cdot，$true $w(V))$ 或 ADP$(r，\cdot，$true $w(V))$。我们遍历所有可行 签名向量

[11] 回顾一下，在同型机器的最大完工时间最小化问题中，输入由一组大小为 s_1、……、s_N的N作业以及一个整数 k组成。目标是将这些作业分配给 k台机器，使最大完工时间最小。这里，最大完工时间指所有 k机器的最大负载。

g 然后取我们所见到的最小值。

我们在以下命题中总结这些算法的保证。注意，对于常数 k、 ϵ、 $\bar{\epsilon}$ 和 W，精确算法的运行时间为 $\tilde{O}(n^4)$，而近似算法的运行时间可简化为 $\tilde{O}(n \cdot h^2)$，其中 h 是˜输入树的高度。因此，对于高度为 $O(1)$ 的树，近似算法非常高效，˜运行时间为 $O(n)$。

▶ **命题 26.** 设 ϵ、 $\bar{\epsilon} > 0$ 和 $k \in \mathbb{N}$。设 $T = (V$、 E、容量) 为一棵有根二叉树，其边权重为 容量(e)，顶点权重为 $w(v) \in \{0,1\}$。那么：

- 精确算法在 T 上用时 $O(M^{2t}n^4)$，为 k-平衡划分问题求得一个双准则 $(1, (1+\bar{\epsilon})(1+\epsilon))$-近似。
- 近似算法在 上用时 $\cdot\log()\log()\log(\log())(k/()))$，为 k T $O\big(nh^2\ M^{2t}\ ^2\ W\ k/\epsilon\ M^t h\ W\ /\epsilon\ /\epsilon^3\big)+$-平衡划分问题求得一个双准则 $(1+\epsilon, (1+\bar{\epsilon})(1+\epsilon))$-近似，其中 $M^t\ \epsilon\bar{\epsilon}\ ^{O(1/\bar{\epsilon}^2}\ h\ T$ 表示 的高度。

证明。 为证明该命题，我们需要论证算法的近似比，并证明划分不会违反平衡约束。我们还需要分析运行时间。

我们首先分析平衡约束。我们证明，在算法返回的解中，连通分量 V_1、.....、 V_k 可以进行划分，使得对于所有 $i = 1$、.....、 k， $w(V_i) \leq (1+\bar{\epsilon})(1+\epsilon)\lceil w(V)/k \rceil$。

考虑 DP 表项 DP(r、 g、true $w(V)$)，其中（真正的）根为 r，连接父节点的边被割去，并且任意属于我们计算出的近似可行签名集合的签名向量 g。那么，这对应于对 $T = T_r$ 的某种划分的代价；在移除割边后， T_r 中的大连通分量 $\mathcal{S}$ 可以与 g 中的表项匹配，使得：

- 一个连通分量 $S \in \mathcal{S}$ 与条目 g_i 匹配，该条目包含 $|S| \leq (1+\epsilon)^i \epsilon \lceil w(V)/k \rceil$，
- 并且恰有 g_i 个连通分量与 g_i匹配。

因此，我们可以如下得到一个划分 $V_1, \ldots, V_k$。首先，我们计算 $I(g)$ 的 $(1+\bar{\epsilon})$-近似解，其中（根据对 g 的假设）最大完工时间至多为 $(1+\bar{\epsilon})(1+\epsilon)\mathrm{d}w(V)/k\mathrm{e}$。这为我们提供了作业到机器的分配。现在，我们将连通分量与作业对应，将集合 V_i 与机器对应，并得到大型组件到 V_i的分配。特别地，每个 V_i 接收的大型组件，其（舍入后的）权重之和至多为 $(1+\bar{\epsilon})(1+\epsilon)\mathrm{d}w(V)/k\mathrm{e}$。现在，我们需要分配算法解中的小型组件。我们可以通过贪心地将每个小型组件（权重小于 $\epsilon\mathrm{d}n/k\mathrm{e}$）分配给当前权重最小的集合 V_i。最后，所有 V_i 的权重都至多为 $(1+\bar{\epsilon})(1+\epsilon)\mathrm{d}w(V)/k\mathrm{e}$（这可由标准论证得出：考虑精确的组件权重时，平均而言，每台服务器的最大完工时间至多为 $w(V)/k$，因此总会有一台最大完工时间至多为$w(V)/k$ 的服务器，当前的小型组件可以分配给它而不违反容量约束）。这意味着，如果算法返回一个目标函数值，那么就存在一个划分 $V_1, \ldots, V_k$，其目标函数值相同且近似可行，即对所有 $i = 1, \ldots, k$，均满足 $w(V_i) \leq (1+\bar{\epsilon})(1+\epsilon)\mathrm{d}w(V)/k$。

接下来，我们来考察这些算法的近似比。考虑使 割$(V_1^*, \ldots, V_k^*)$ 最小的最优划分 OPT $= (V_1^*, \ldots, V_k^*)$，满足对于所有 i， $w(V_i^*) \leq \lceil w(V)/k \rceil$。我们首先说明，OPT 会在精确动态规划中产生一个具有可行签名且代价为 OPT 的动态规划表项。为此，取最优划分 $V_1^*, \ldots, V_k^*$，并将每个大型连通分量的权重向上舍入到下一个形如 $(1+\epsilon)^i \cdot \epsilon\lceil n/k \rceil$ 的值。令 $g = (g_0, \ldots, g_{t-1}) \in [M-1]^t$ 为签名，其中 g_i 表示

OPT 中舍入权重为 $(1+\epsilon)^i \cdot \epsilon\lceil n/k\rceil$ 的大型组件数量。注意，由于对于所有 i，$w(V_i^*) \le \lceil w(V)/k\rceil$，因此 V_i^* 中组件的总舍入权重至多为 $(1+\epsilon)\cdot\lceil w(V)/k\rceil$，因为组件权重至多增加 $(1+\epsilon)$ 倍。因此，构造出的签名向量 g 是可行的，因为划分 $V_1^*,\dots,V_k^*$ 会产生 $I(g)$ 的可行解。此外，舍入不会影响目标函数值，因此 OPT 会在精确动态规划中产生一个具有可行签名且代价为 OPT 的动态规划表项。这意味着精确动态规划的最优值至多为 割$(V_1^*,\dots,V_k^*)$。结合上述关于动态规划近似满足平衡约束的结论，我们得到精确算法计算出一个双准则 $(1,(1+\bar\epsilon)(1+\epsilon))$-近似。

现在让我们转向第 D.2 节中近似算法的近似比。回想一下，根据引理 24，精确动态规划是良态的；而在引理 25 中，我们证明了在每一步中，近似算法在树 T 的每个层级上至多损失因子 $1+\delta$。现在，我们可以应用引理 19，得出根节点处的近似为 $(1+\delta)^{h+1}$，其中 h 是树 T 的高度。因此，如果我们设置 $\delta = \ln(1+\epsilon)/(h+1)$，那么近似动态规划的近似比为 $1+\epsilon$，因为此时 $(1+\delta)^{h+1} \le \exp(\delta(h+1)) = 1+\epsilon$。由于引理 19 中的近似概念适用于所有形如 ADP$(r,\ g,$ 真，$\cdot)$ 的函数以及 x 的所有可能取值，因此我们得到，该近似算法计算出一个双准则 $(1+\epsilon,\ (1+\bar\epsilon)(1+\epsilon))$-近似。

我们通过考察算法的运行时间来完成命题的证明。注意，就运行时间而言，两种算法的区别仅在于填充动态规划单元所需的时间，而计算解所需的时间相同。

我们先考虑按照 D.3 节计算解所需的时间。首先，考虑求解多项式时间近似方案所需的时间，即 $(N/\bar\epsilon)^{O(1/\bar\epsilon^2)}$，其中 N 表示作业总数。注意，在我们的情形中，作业数至多为 $N \le k(1+1/\epsilon)$：每个作业的大小至少为 $\epsilon\lceil n/k\rceil$，因此在最优解中，一台机器最多可容纳 $1+1/\epsilon$ 个作业。因此，如果作业数超过 $k(1+1/\epsilon)$，多项式时间近似方案就可以直接拒绝该实例并判定其不可行。由此，单次运行该多项式时间近似方案所需的时间为 $(k/(\epsilon\bar\epsilon))^{O(1/\bar\epsilon^2)}$。由于我们必须针对 M^t 个签名分别运行该多项式时间近似方案，因此寻找近似可行配置所需的总时间为 $M^t(k/(\epsilon\bar\epsilon))^{O(1/\bar\epsilon^2)}$。

最后，我们来考虑填充动态规划单元所需的时间。对于精确动态规划，引理 24 表明，填充一个动态规划单元 DP$(v,\cdot,\cdot,\cdot)$ 需要 $O(M^{2t}n^3)$ 的时间。然后，通过应用引理 19，计算所有动态规划单元的总时间为 $O(M^{2t}n^4)$。对于近似动态规划，根据引理 25，填充单个动态规划单元 ADP$(v,g,$ 割$,\cdot)$ 需要 $O(M^t p^2 \log(M^t p)\log(k/\epsilon)/\epsilon))$ 的时间。由于 g 有 M^t 种选择，再次应用引理 19 可得，填充近似动态规划表的总运行时间为 $O(nM^{2t}p^2\log(M^t p)\log(k/\epsilon)/\epsilon))$。由于在 D.2 节中我们选取分段数为 $p = O(\log_{1+\delta}(W))$，并且在上文中选取了 $\delta = O(\epsilon/h)$，因此运行时间的上界为 $O\big(nM^{2t}\cdot\big(1/\epsilon\cdot h\log W\big)^2\cdot\log(k/\epsilon)/\epsilon\cdot\log(M^t h\log(W)/\epsilon)\big) = O\big(nh^2\cdot M^{2t}\log^2(W)\log(k/\epsilon)\log(M^t h\log(W)/\epsilon)/\epsilon^3\big)$。◀

D.4 推广到一般图

现在，我们将命题 26 的结果从二叉树推广到一般图。

我们首先将其推广到一般图，并在其中使用 Räcke 树（参见 C.1节）。由于 Räcke 树可能不是二叉的，我们现在引入二叉化*Räcke*树 这一概念，它本质上描述了如何将一棵非二叉 Räcke 树转化为一棵与 Räcke 树非常相似的二叉树。稍后，二叉化Räcke树将使我们能够对其应用命题 26。

▶ **定义 27**（二叉化Räcke树）。设 $G=$（V_G, E_G,容量$_G$）为一个加权图。若加权有根树 $T=$（V_T, E_T, 容量$_T$）满足以下性质，则称其为 G 的二叉化Räcke树：

- T 是一棵有根二叉树。$V_G \subseteq V_T$。T 中的所有边的权重均属于 W_∞。令 T' 表示在 T 中
- 收缩所有权重为 ∞ 的边后得到的树。则 T' 是 G的一棵*Räcke*树。
-
-

我们将上一要点中的树 T' 称为T的对应的（非二叉化）Räcke树。如果对应的*Räcke*树 T' 具有质量 q，则称 T具有质量 q。

接下来，我们注意到，T 中代价有限的每个割都对应于相应Räcke树 T' 中的一个割，反之亦然。因此，T' 中代价有限的割近似于初始图 G的割结构。我们在下面的观察中对此进行更形式化的表述。

▶ **观察 28**。令 $G=(V_G, E_G,$ 容量$_G)$ 为一个加权图，并令 $T=(V_T, E_T,$ 容量$_T)$为 G 的一棵质量为 q的二叉化*Räcke*树。则对于所有不相交子集 A、$B \subseteq V_G$，均有最小割$_G(A, B) \leq$最小割$_T(A, B) \leq q \cdot$最小割$_G(A, B)$。

证明。令 T' 为 T所对应的（未二叉化）Räcke 树，并考虑两个不相交的顶点子集 A, $B \subseteq V_G$。我们证明 $\min_T(A, B) = \text{mincut}_{T'}(A, B)$。由于 T 的质量为 q（根据假设），因此 T' 是 G 的一棵质量为 q 的 Räcke 树，满足上述观察中的性质，故结论立即成立。

由于 T'只能通过收缩边从 T得到，因此 $\text{mincut}_{T'}(A, B) \geq \text{mincut}_T(A, B)$。接下来，我们证明 $\text{mincut}_T(A, B) \geq \text{mincut}_{T'}(A, B)$。首先注意，$\text{mincut}_{T'}(A, B) \leq q \cdot \text{mincut}(A, B) < \infty$。从 T变为 T'时，我们只收缩权重为 ∞ 的边，因此 T不包含任何不属于 T'且代价有限的割。因此，$\text{mincut}_T(A, B) \geq \text{mincut}_{T'}(A, B)$。◀

此外，我们证明了可以在近线性时间内计算出一棵质量良好的二叉化Räcke树。

▶ **引理 29**。设 $G=(V_G, E_G,$容量$_G)$ 是一个具有 n 个顶点和 m 条边的加权图。我们可以在 $\tilde{O}(m)$ 时间内计算出一棵二叉化*Räcke*树 $T=(V_T, E_T,$ 容量$_T)$，它有 $O(n)$ 个顶点，高度为 $O(\log^2 n)$，质量为 $O(\log^4 n)$。

证明。设 $T'=(V_{T'}, E_{T'},$ 容量$_{T'})$ 是定理 15 中针对 G 的 Räcke 树，可以在时间 $\tilde{O}(m)$ 内计算得到。首先注意，T'有 $n_{T'} := O(n)$ 个顶点，高度为 $O(\log n)$。其次，注意到 T' 的度可能没有上界。因此，我们将说明如何计算一棵二叉化Räcke树 T，使其对应的（非二叉化）Räcke树为 T'。具体做法是，将 T' 中的每个顶点 u 替换为一棵平衡二叉树 τ_u，其叶节点数为 $\deg(u)$，其中 $\deg(u)$ 表示 u的子节点数。τ_u 的内部边的权重为 ∞；在 T中连接子树 τ_u 和 $\tau_v, u \neq v$ 的边，对应于 T' 中的边，并且其权重与 T'中的相同（且有限）。我们将看到，通过收缩 T中所有权重为 ∞的边，就能得到 T'。下面详细说明这一过程。

我们按如下方式构造 T。首先，根据定理 15 中的算法计算 T'。现在按如下方式构造 T。对于每个顶点 $u \in V_{T'}$，我们添加一棵有 $\deg(u)$ 个叶节点的平衡有根二叉树 τ_u。我们将 τ_u 的根记为 r_u。我们将 τ_u 的每个叶节点与 u 的一个子节点对应起来，并将与子节点 v 对应的 τ_u 的叶节点记为 $c_{u, v}$。我们将 τ_u 内部边的权重设为 ∞。注意，对于每个顶点 u，树 τ_u 有 $O(\deg(u))$ 个顶点，因此 T 有 $O(n_{T'}) = O(n)$ 个顶点。接下来，对于每条边 $(u, v) \in E_{T'}$（这里我们假设

v 是 u 的子节点），我们在 T 中插入边 $(c_{u,v}, r_v)$，并将 $\text{cap}_T(c_{u,v}, r_u)$ 设为 $\text{cap}_{T'}(u, v)$。最后，如果 u 是 T' 的根，那么我们将 r_u 设为 T 的根。

接下来只需证明 T 是高度为 $O(\log^2 n)$、质量为 $O(\log^4 n)$ 的二叉化Räcke树。显然，T 是一棵二叉树，因为每个子树 τ_u 中的所有顶点都至多有两个子节点；此外，每个顶点 $c_{u,v}$ 都至多有一个子节点（即 r_v）。接下来，由于 T' 的高度为 $O(\log n)$，且子树 τ_u 的高度为 $O(\log n)$，因此 T的高度为 $O(\log^2 n)$。最后，令 T'' 为通过收缩所有权重为 ∞ 的边而从 T 得到的树。我们证明 $T' = T''$。事实上，考虑 T中的任意顶点 $u \in V_{T'}$及其子树 τ_u。收缩 τ_u中的边后，剩下的子树仅包含 r_u。此外，不同子树 τ_u 和 τ_v, $u \neq v$中的顶点之间的所有边权重有限。因此，$T' = T''$。这意味着 T是 G的二叉化Räcke树。由于 T' 的质量为 $O(\log^4 n)$，因此 T 的质量也为 $O(\log^4 n)$。◀

我们通过证明定理 2 来结束本小节。

定理 2 的证明。关于一般图的断言可按如下方式证明。设 $G = (V_G, E_G, \text{cap}_G)$ 是一个有 n 个顶点的加权图。根据引理 29，我们可以在 $O(n)$ 时间内计算一个二叉化的 Räcke̊树 $T = (V_T, E_T, \text{cap}_T)$，它有 $O(n)$ 个顶点。在 T 中，我们给所有顶点 $v \in V_G \cap V_T$（即 T 中对应于 G 中顶点的叶节点）赋予权重 $w(v) = 1$，并给所有顶点 $v \in V_T \setminus V_G$（即 T 中不对应于 G 中任何顶点的内部节点）赋予权重 $w(v) = 0$。现在注意到 $w(V) = n$，因此 $V_1, \ldots, V_k$ 对 T 的一个平衡划分（其中 $w(V_i) \leq (1+\epsilon)\lceil w(V)/k \rceil$ 对所有 i 均成立），对应于 $V_1', \ldots, V_k'$ 对 G 的一个平衡划分，其中对所有 i，$|V_i'| \leq (1+\epsilon)\lceil n/k \rceil$，而 $V_i' = \{v \in V_i : w(v) = 1\}$。现在结合观察 28、命题 26，以及 T 的质量为 $O(\log^4 n)$ 这一事实，即可得到所述断言。

为了得到关于一般树的结果 T'（其度数无界），我们采用类似的方法。我们完全按照引理 29 的证明构造一棵二叉化树 T。现在，在 T 中，我们对所有子树 τ_v 的根顶点设定$w(r_v) = 1$，并对这些子树中的所有其他顶点设定 $w(v) = 0$ τ_v。与之前类似，注意到 $w(V_T) = n$，因此，T的一个平衡划分$V_1, \ldots, V_k$，满足对所有 i都有 $w(V_i) \leq (1+\epsilon)\lceil w(V)/k \rceil$，对应于 T'的一个平衡划分$V_1', \ldots, V_k'$，满足对所有 i都有 $|V_i'| \leq (1+\epsilon)\lceil n/k \rceil$，其中 $V_i' = \{v \in V_{T'} : r_v \in V_i\}$。于是，根据命题 26，便可得到无界度数树的证明。◀

D.5 扩展到动态设置

接下来，我们将介绍新的动态算法，其中边会从图中插入和删除。我们将给出适用于树和一般图的新算法。

扩展到动态树。我们先讨论 T 为一棵经历边插入与删除的二叉树时的情形。我们将使用引理 20，使命题 26 中的结果具有动态性。不过，这里存在一个小小的技术难点：由于边删除，T 会变成一片森林，并分裂成多个连通分量。当插入一条边 (u, v)，且 u 和 v 在各自的连通分量中都已有父节点时，就会出现问题。在这种情况下，我们无法立即将 u 设为 v 的根节点（反之亦然）。因此，我们需要找到一种高效的方法，对包含 v的树重新定根；也就是说，我们需要将 v 设为其所在连通分量的根节点，并确保无需重新计算 v所在连通分量中所有顶点的动态规划解。下面我们将更详细地描述我们的动态算法。

首先，假设从 T 中删除一条边 (u, v)，并且假设在删除该边之前，u 比 v更靠近 T 的根。此时，T 变成由多个连通分量组成的森林。在这种情况下，我们将 v 设为其所在连通分量的根，并重新计算动态规划

解（由于 v 没有父节点，我们只需重新计算 v 的动态规划单元）。此外，对于 u 及其所有祖先，我们按照引理 20 重新计算动态规划解。

接下来，假设插入一条边 (u, v)，其中 u 和 v 位于不同的连通分量中。进一步假设插入这条边后，u 是 v 的父节点。接下来分两种情况讨论：v 是否为其所在连通分量的根节点。

首先，假设 v 是其所在连通分量的根节点。那么，我们只需将边 (u, v) 插入 T，并按照引理 20 重新计算 v 及其所有祖先（包括 u）的解。

其次，假设 u 和 v 都不是其所在连通分量的根节点。此时，我们首先必须对包含 v 的连通分量重新指定根节点，使 v 成为其根节点，并确保所有动态规划解都有效。具体做法如下。令 $v = v_1$、……、v_ℓ 表示从 v 到其所在连通分量的根节点 v_ℓ 的路径上的顶点（在插入边之前）。然后，我们按照上述边删除例程，依次从 T 中删除所有边 $(v_\ell, v_{\ell-1})$、……、(v_2, v_1)。注意，删除后，这些 v_i 都没有父节点，因此每个 v_i 都是其所在连通分量的根节点。此外，根据我们选择边删除顺序的方式，第 i 次删除后，我们只需重新计算顶点 $v_{\ell-i}$ 和 $v_{\ell-i-1}$ 的动态规划单元。接下来，我们以相反的方向重新插入这些边，也就是说，按照此顺序插入边 $(v_{\ell-1}, v_\ell)$、……、(v_1, v_2)。这样，$v = v_1$ 就成为该连通分量的根节点。插入这些边时，我们使用上一段中的子程序，并利用每个 v_i 都是其所在连通分量的父节点这一点，从而高效地更新动态规划解：根据我们选择边插入顺序的方式，第 i 次边插入后，我们只需重新计算顶点 $v_{\ell-i-1}$ 和 $v_{\ell-i}$ 的动态规划单元。完成包含 v 的连通分量的重新平衡后，v 已成为其所在连通分量的父节点，因此我们可以使用上述例程插入边 (u, v)。至此，边插入过程完成。

接下来，当我们想要输出动态规划解的值时，只需使用 D.3 节中描述的子程序。

我们在下面的命题中总结动态算法的保证。注意，当参数 ϵ、$\bar{\epsilon}$、k 和 W 为常数时，更新时间变为 $\tilde{O}(h^3)$，查询时间则为 $O(1)$。因此，对于高度相对于顶点数为多对数或次多项式的树，该算法非常高效。

▶ 命题 *30.* 设 ϵ、$\bar{\epsilon} > 0$ 和 $k \in \mathbb{N}$。设 $T = (V, E,$ 容量$)$ 为一棵有根二叉树，其边权重 $cap(e) \in W_\infty$，顶点权重为 $w(v) \in \{0,1\}$，并且正在进行边插入与删除。设 h 为该树在任意时刻的高度上界 T。那么，存在一种全动态算法，能够维护 T 上 k-平衡划分问题的双准则 $(1+\epsilon, (1+\bar{\epsilon})(1+\epsilon))$-近似，其更新时间为 $O\big(h^3 \cdot M^{2t} \log^2(W) \log(k/\epsilon) \log(M^t h \log(W)/\epsilon)/\epsilon^3\big)$，查询时间为 $M^t(k/(\epsilon\bar{\epsilon}))^{O(1/\bar{\epsilon}^2)}$。

证明。 算法维持双准则 $(1+\epsilon, (1+\bar{\epsilon})(1+\epsilon))$-近似这一事实，可由引理 20 以及命题 26 的证明中相同的论证立即得出；在该证明中，我们说明了近似动态规划满足引理 19 的条件。

接下来需要分析更新和查询时间。对于查询时间，注意我们所做的只是运行 D.3 节中的子程序。正如我们在命题 26 的证明中所论证的，该子程序的运行时间为 $M^t(k/(\epsilon\bar{\epsilon}))^{O(1/\bar{\epsilon}^2)}$。这就证明了关于查询时间的结论。

对于更新时间，我们先考虑边删除（u，v）。此时，我们需要更新 v 的动态规划单元，以及 u 和它所有祖先的动态规划解。根据引理 20、引理 25，以及我们对 $p = O(h \log W/\epsilon)$ 的选择，这可以在 $O\big(h^3 \cdot M^{2t} \log^2(W) \log(k/\epsilon) \log(M^t h \log(W)/\epsilon)/\epsilon^3\big)$ 的时间内完成。

接下来，考虑插入（u, v）且 v 是其连通分量的根节点的情况。此时，我们需要重新计算 v 及其所有祖先（包括 u）的动态规划解。根据引理 20、引理 25，以及我们对 $p = O$（$h$log W/ϵ）的选择，这可以在引理所述的时间内完成。在需要对 v所在的连通分量重新定根时，注意到我们必须重新计算 v所有祖先的解。由于 T 的高度有 h的上界，此类祖先至多有 h 个。此外，我们选择边删除的顺序，使得在重新定根过程中，每当删除或插入一条边时，只需重新计算两个动态规划单元。因此，重新定根过程中，我们总共只需重新计算 O(ℎ)个动态规划单元的解；由引理 25 可知，这一过程的总时间为 $O\big(h^3 \cdot M^{2t}\log^2(W)\log(k/\epsilon)\log(M^t h\log(W)/\epsilon)/\epsilon^3\big)$。◀

扩展至动态一般图和非二叉树。 现在，假设我们的输入是一个正在经历边插入与删除的动态（一般）图 G 。本质上，我们将通过维护一棵动态 Räcke 树，并在其上运行命题 30 中的算法来解决这个问题。然而，定理 16 中的动态 Räcke 树是非二叉的，因此我们首先在以下引理中论证，我们可以动态维护一棵二叉化 Räcke 树。

▶ **引理 31。** 设 $G = (V_G, E_G)$ 是一个包含 n 个顶点且正在经历边插入与删除的动态无权图。我们可以维护一棵包含 $O(n^2)$ 个顶点、高度为 $O(\log^{7/6} n)$ 且质量为 $n^{o(1)}$ 的二叉化 *Räcke* 树 T =(V_T, E_T, 容量$_T$)，摊销更新时间为 $n^{o(1)}$。预处理时间为 $O(n^2)$。

证明。令 T' =($V_{T'}$, $E_{T'}$, 容量$_{T'}$) 为定理 16 中的 G 所对应的全动态Räcke树。首先，注意到 T' 有 $n_{T'}$:= O(n) 个顶点，高度为 $O(\log^{1/6} n)$。其次，注意到 T' 的度可能无界。因此，类似于引理 29 的证明，我们将展示如何维护一棵二叉化Räcke树 T，使其对应的（非二叉化）Räcke树为 T' 。具体做法是取 T'，并将 T' 中的每个顶点 u 替换为一棵平衡二叉树τ_u ，该树有 $n_{T'}$ 个叶节点； τ_u 的内部边权重为 ∞ ，而在 T 中连接子树 τ_u 和 τ_v, $u \neq v$ 的边，则对应于 T' 中的边，并具有与 T' 中相同的（有限）权重。我们将看到，在 T 中收缩所有权重为 ∞ 的边后，将再次得到 T'。下面详细说明这一过程。

在预处理过程中，我们首先构建 T'。注意，这需要 O(n^2) 时间。现在我们按如下方式构造 T 。对于每个顶点 $u \in V_{T'}$，我们添加一棵具有 $n_{T'}$ 个叶节点的平衡有根二叉树τ_u 。我们将 τ_u 的根节点称为 r_u。我们将 τ_u 的每个叶节点与一个顶点 $v \in V_{T'}$ 对应起来，并将 τ_u 中与 v 对应的叶节点记为 $c_{u,v}$。我们将 τ_u 内部各边的权重设为 ∞。注意， T 有 $O(n_{T'}^2)$ 个 $= O(n^2)$ 顶点。接下来，对于每条边 $(u, v) \in E_{T'}$（这里假设 v 是 u 的子节点），我们在 T 中插入边 $(c_{u,v}, r_v)$，并设置 cap$_T$ $(c_{u,v}, r_u)$ =cap$_{T'}(u, v)$。最后，如果 u 是 T' 的根节点，那么我们将 r_u 设为 T 的根节点。

接下来，假设 G 因边插入或删除而发生变化。然后，我们首先使用定理 16 中的算法更新树 T'。现在，每当在 T' 中插入（删除）一条边 (u, v) 时，我们就将边 $(c_{u,v}, r_v)$ 插入到 T 中（从 T 中删除）。在 T 中进行上述每次插入和删除的时间均为 O(1)。由于更新 T' （通过定理 16）需要 $n^{o(1)}$ 的摊销时间，因此总更新时间为 $n^{o(1)}$。

尚需证明， T 是高度为 $O(\log^{7/6} n)$、质量为 $n^{o(1)}$的二叉化Räcke树。显然， T 是一棵二叉树，因为每个子树 τ_u 中的所有顶点至多有两个子节点；此外，每个顶点 $c_{u,\ v}$ 至多有一个子节点（即 r_v）。接下来，T的高度为 $O(\log^{7/6} n)$，因为 T' 的高度为 $O(\log^{1/6} n)$，且子树 τ_u 的高度为$O(\log\ n)$。最后，令 T'' 为将所有权重为∞的边收缩后由 T得到的树。我们证明 $T' = T''$。事实上，考虑任意顶点 $u \in V_{T'}$及其在 T中的子树 τ_u。

然后，收缩 τ_u中的边后，我们得到的子树只包含 r_u。此外，不同子树 τ_u 和 τ_v之间的所有边 $u \neq v$都具有有限权重。因此， $T' = T''$。这意味着， T是 G的二叉化Räcke树。由于 T' 的质量为 $n^{o(1)}$， T 的质量也为 $n^{o(1)}$。◀

根据上述引理，我们针对动态一般图的动态算法 G 如下。我们按照引理 31，在输入图 G上维护动态二叉化 Räcke 树 T；也就是说，每当在 G中插入或删除一条边时，我们也会更新引理中的数据结构。请注意，这也会导致在 T 中插入和删除边。和之前一样，我们设置 T 中的顶点权重，使得如果 v 对应于G 中的一个顶点，则 $w(v) = 1$；如果 v 是 T 中不对应于 G中任何顶点的内部节点，则 $w(v) = 0$。此外，我们在 T上运行命题 30 中针对二叉树的动态算法。具体而言，每当 T 更新时，我们也按照命题 30 更新动态规划解。我们还使用该命题中的相同查询过程。

W我们通过证明定理来结束本小节 m 3.

定理 3 的证明。我们首先证明一般图的结果。由于我们维护的动态二叉化 Räcke 树 T 的质量为 $n^{o(1)}$，定理 2 证明中的相同论证表明，我们维护的是一个双准则$n^{o(1)},(1+\bar{\epsilon})(1+\epsilon)$-近似 G。根据引理 31，我们可以维护 T，其摊销更新时间为 $n^{o(1)}$；因此，每次更新操作中，向 T 插入和删除边的摊销次数为 $n^{o(1)}$。由于T 的高度为 $O(\log^{7/6} n)$，且根据命题 30，总摊销更新时间 $n^{o(1)}$。这就证明了关于动态一般图的结论。

为了得到关于非二叉树的结果，我们可以采用与上述类似的方法。考虑一棵正在进行边插入与删除操作的非二叉 T'。我们可以像引理 31 的证明中那样维护相同的数据结构，从而得到一棵含有 $O(n^2)$ 个顶点、最坏情况更新时间为 $O(1)$ 的二叉树 T。现在，我们为 T 中作为子树 τ_v 根节点的所有顶点 r_v 赋予权重 $w(r_v) = 1$，并为这些子树 τ_v 中的所有其他顶点赋予权重 $w(v) = 0$。根据定理 2 证明中的相同论证，即可得到所述结论。◀

E 同时源定位

本节将介绍由 Andreev 等人 [4] 研究的同时源位置问题的高效算法。回顾一下，该问题的输入包括一个图$G = (V, E,$ 容量 $d)$，图的边上有容量函数容量： $E \to W_\infty$，顶点上有需求函数 d： $V \to W_\infty$。目标是选取最小集合$S \subseteq V$，其中包含能够同时满足所有顶点需求的源点。更具体地说，如果存在一股从集合 S 中的顶点流出的流，能够向所有顶点 $v \in V$ 供应需求 $d(v)$，且不违反边上的容量约束，那么源点集合 S 就是可行的。这里假设每个源顶点都可以发送任意多的流量，唯一限制是边容量。目标是找到规模最小的可行源点集合。

接下来，我们总结同时源位置问题的主要结果。首先，我们介绍双准则近似的概念。设 S^* 为同时源位置问题的最优解。若解 S 满足 $|S| \leq \alpha |S^*|$，且在所有边容量都扩大 β 倍后， S 是一个可行的源点集合，则称其为双准则（α， β）近似解。

以下定理概述了我们关于静态算法的主要结果。

▶ 定理 4。设 $\epsilon > 0$。设 $G = (V, E,$ 容量 $d)$ 是一个具有n 个顶点和 m 条边的无向加权图。那么，对于同时源位置问题，我们可以计算：一个 $(1+\epsilon, O(\log^4(n)))$-近似，时间复杂度为[12] $\tilde{O}(\frac{1}{\epsilon^2}m)$。如果 G 是一棵高度为 h 的树，则可在时间 $\tilde{O}(\frac{1}{\epsilon^2}h^2 \cdot n)$ 内计算一个 $(1+\epsilon, 1)$-近似。

接下来，我们介绍支持以下更新操作的动态算法：设置需求(v, d)：将顶点 v 的需求更新为 $d(v) = d$，设置容量$((u, v), c)$：将边 (u, v) 的容量更新为容量$(u, v) = c$，删除(u, v)：从图中删除边 (u, v)，插入 $((u, v), c)$：将边 (u, v) 插入图中，其容量为容量$(u, v) = c$。下一个定理总结了我们关于动态算法的主要结果。

▶ **定理 5.** 设 $\epsilon > 0$。设 $G = (V, E,$ 容量 $d)$ 是一个具有 n 个顶点和 m 条边、并执行上述更新操作的图。那么，对于同时源位置问题，我们可以维护：

- 若所有边容量均为 1，则可获得一个 $(1+\epsilon, n^{o(1)})$-近似解，摊销更新时间为 $n^{o(1)}/\epsilon^2$，预处理时间为$O(n^2/\epsilon^2)$。若只允许更新操作设置需求(v, d)，则可获得一个 $(1+\epsilon, O(\log^4(n)))$-近似解，最坏情况更新时间为 $\tilde{O}(1/\epsilon^2)$，预处理时间为 $\tilde{O}(m)$。若只允许更新操作设置需求(v, d)，则可获得一个 $(1+\epsilon, O(\log^2(n) \log\log(n)))$-近似解，最坏情况更新时间为 $\tilde{O}(1/\epsilon^2)$，预处理时间为 $poly(n)$。若 G 是一棵高度为 h 的树，则可获得一个 $(1+\epsilon, 1)$- 近似解，最坏情况更新时间为 $\tilde{O}(h^3/\epsilon^2)$，预处理时间为 $O(n^2/\epsilon^2)$。

请注意，在我们的静态和动态算法中，我们可以像在背包问题的命题 12 后所描述的那样，输出相应的解。

我们首先在 E.1 节介绍二叉树这一特殊情形的精确动态规划，然后在 E.2 节介绍近似动态规划。之后，我们在 E.3 节将结果从二叉树推广到一般图，并在 E.4 节进一步推广到全动态设置。

E.1 ExactDP

我们考虑二叉树上的同时源位置问题这一特殊情形，并给出一个能够精确求解该问题的动态规划。我们用 $T = (V, E,$ 容量 $d)$ 表示作为输入的、根节点为 r 的有根二叉树。此外，我们假设对于每个顶点$v \in V$，输入还会告知我们是否允许将其设为 v 源点；请注意，这只是对问题的推广（因为在原问题中，所有顶点都可以设为源点）。稍后在 E.3 节中，这一推广会很有帮助，因为应用 Räcke 树时，我们只希望允许叶节点充当源点。

E.1.1 动态规划定义

现在我们定义精确动态规划。我们还将讨论它与 Andreev 等人的动态规划之间的关系，以及为何我们没有采用 Andreev 等人的动态规划。给定一个顶点 v 和一个值$x \in \mathbb{R}$，我们将 DP(v, x) 记为在 T_v 中放置源点的最小数量，使得当 v 从其父节点接收的流不超过 x 时，能够满足 T_v 中的所有需求。注意，x 可以取正值或负值：对于 $x \geq 0$，这对应于流从父节点

[12] 我们用 $\tilde{O}(f(n, \epsilon, W))$ 表示形如 $f(n, \epsilon, W) \cdot \text{polylog}(n, \epsilon, \log W)$ 的运行时间。

流入 T_v 的情形；对于 $x < 0$ ，这对应于流从 T_v 流向 v的父节点的情形。我们进一步遵循以下约定：当 v 从其父节点接收流 x 时，若无法满足 T_v 中的需求，则令 DP(v, x) $= \infty$。

注意，该动态规划有行 $\mathcal{I} = V$ 和列 $\mathcal{J} = \mathbb{R}$。我们将使用第 2 节中的数据结构，利用单调分段常数函数存储行动态规划(v,·)。接下来，我们注意到每个动态规划(v,·)都是单调递减的。因此，该动态规划满足定义8的性质（1）。

▶ **观察32**。 函数 动态规划$(v, \cdot)$: $\mathbb{R} \to [n+1] \cup \{\infty\}$ 单调递减。

证明。由动态规划(v, x)的定义可立即得出：考虑 $x, x' \in \mathbb{R}$ with$x \le x'$。那么，若 T_v 从 v 的父节点接收的流不超过 x ，则相应的解在 T_v 从 v 的父节点接收的流不超过 x' 时也可行。因此，动态规划(v, x) $\ge$ 动态规划(v, x')，证明完毕。◀

注意，对于 T 上的同时源位置问题，其全局解可通过计算动态规划(r, 0)获得，其中 r 是 T的根节点：首先， r 没有父节点，因此它必须自身是一个源点，或者由其子节点满足其需求；这说明了选择 $x = 0$的原因。此外，根据定义，动态规划(r,0)是满足 $T_r = T$ 中所有需求所需的最少源点数，因此我们得到的流是可行的。由此可知，动态规划(r,0)给出了全局最优解。

与 Andreev 等人方法的关系 [4]。接下来，我们详细阐述我们的动态规划与 Andreev 等人 [4] 所使用的函数 f 之间的关系。在 [4], 中，动态规划计算的函数 f 定义如下。给定一个顶点 v 和一个整数 $i \in \mathbb{N}$，Andreev 等人定义函数 $f(v, i)$，表示在满足 T_v 中所有需求且可以在子树 T_v 中放置 i 个源点的情况下， v 需要从其父节点接收的最小流量。与上文类似， $f(v, i)$ 可以取正值或负值：如果只能通过从父节点接收流量才能满足 T_v 中的需求，那么 $f(v, i)$ 为正；如果子树 T_v 中的源点已经满足了 T_v 中的需求，那么 v 可能会向其父节点发送流量，此时 $f(v, i)$ 为负。不难看出，函数 $f(v, i)$ 关于 i 单调递减。[13]

现在将 $f(v, \cdot)$: $\mathbb{N} \to \mathbb{R}$ 视为一个函数，并考虑其“逆”[14] 函数 $f^{-1}(v, \cdot)$: $\mathbb{R} \to \mathbb{N}$，其中 f^{-1} 定义在整个实数集上（包括负数）。也就是说， $f^{-1}(v, x)$ 表示我们需要在 T_v 中放置的源点的最小数量，使得 v 向其父节点提出的需求至多为 x。但这正是 DP(v, x) 的定义。因此，对于所有 v和 x，DP(v, x) $= f^{-1}(v, x)$。

为什么我们没有使用 f? 在 [4] 中，文中展示了如何通过自底向上动态规划，在每个动态规划单元中仅使用少量分类讨论和一次（min $+$）卷积，在多项式时间内计算函数 f 。因此，我们可能会好奇，为什么我们为动态规划选择了 DP($v, \cdot$) $= f^{-1}(v, \cdot)$，而不是 f? 事实上，将函数 f 视为单调分段常数函数，并将其用于动态规划，似乎是很自然的做法。

---

[13] 这直接由 f的定义以及以下事实得出：向子树 T_v中添加更多源点后， T_v 需要从 v的父节点接收的流量只会减少。[14] 我们注意到，严格来说， $f(v, \cdot)$ 没有逆函数，因为可能有多个值映射到同一个数，即对于 $i \neq i'$， $f(v, i) = f(v, i')$。因此，严格来说，我们定义 $f^{-1}(v, x) = \min\{i\colon\ f(v, i) \le x\}$，其中我们遵循约定 $\min\{\emptyset\} = \infty$。随后，我们将 $f^{-1}(v, \cdot)$ 解释为从 $\mathbb{R}$ 到 $[n+1]$的分段常数函数。

对于精确计算的情况，这种做法是可行的；而我们现在简要说明，为什么它在后文讨论的近似情形中似乎不太方便。

假设我们在近似计算中使用函数 f。为了得到高效的近似算法，我们必须确保 f 只有少量分段，而实现这一点的主要方法是按照引理 6 对 f 进行舍入。然而，这会变得棘手，因为 f 的函数值既有正值也有负值。接下来，将 f 的正函数值视为需要满足的顶点需求，将 f 的负函数值视为可用边容量，会有助于理解。主要问题在于，由于 f 的函数值有正有负，因此舍入方式并不明确：如果将正值和负值都向上舍入（朝 $+\infty$ 的方向），就相当于增加顶点需求，同时减少边容量；但这可能会使某些可行解（在精确计算中）在舍入计算中变得不可行。另一方面，可以设想，若始终将 f向下舍入（朝 $-\infty$的方向），本质上就会减少顶点需求，同时增加边容量。如果允许边容量违反一个 $(1+\epsilon)$ 倍数，这种方法或许可行。不过，即便如此，我们还会遇到另一个问题：为了用少量分段表示 f，对于 $[-1,1]$ 中的函数值和 $[-W,W]\setminus[-1,1]$中的函数值，我们必须采用不同的舍入机制，其中 W 是最大边容量。事实上，如果将 f 的值舍入到 $(1+\delta)^j$ 的幂，那么 $[-W,W]\setminus[-1,1]$ 中只有 $O(\log_{1+\delta}(W))$ 个函数值，但 $[-1,1]$中却有无穷多个函数值。类似地，如果舍入到 δ 的倍数，那么 $[-1,1]$ 中只有 $O(1/\delta)$ 个函数值，但这会导致 $[-W,W]\setminus[-1,1]$ 中出现 $O(W/\delta)$ 个函数值。无论哪种情况，我们的函数都会有过多的分段，因此必须选择一个能在这两种情况之间取得权衡的舍入函数。此外，我们还必须进行分析，证明这种“更复杂”的舍入函数不会引入过多误差。

请注意，在上述讨论中，所有问题都源于 $f(v,\cdot)$ 也可能取负值。另一方面，我们的动态规划（即 $f^{-1}(v,\cdot)$）只取非负函数值，因此我们可以使用将数值舍入到 $1+\delta$ 的幂的标准舍入函数 $\lceil\cdot\rceil_{1+\delta}$，从而避免上述所有复杂情况。因此，我们绕开了上述所有问题。

我们的方法还有一个积极的副作用：我们的运行时间中不再出现 $\mathrm{polylog}(W)$ 因子，而只出现 $\mathrm{polylog}(n)$ 因子，因为单调分段常数函数的陪域变成了 $[n+1]$，而不是某个可能很大的区间$[-W,W]$。

E.1.2 计算动态规划

现在我们描述如何精确计算动态规划。由此将引出定义 8 中的过程 $\mathcal{P}_i$。

设 $v\in V$ 是 T 中的任意一个顶点。假设我们已经计算出 v 的子节点（如果存在）的解，我们将说明如何高效计算 $\mathrm{DP}(v,\cdot)$。请回想，对于每个顶点 v，我们还会将 v 是否可以用作源点作为输入。在接下来的分类讨论中，每当我们考虑将 v 用作源点的情况时，都会默认以 v 也可以用作源点为前提；如果 v 不能用作源点，我们就跳过这种情况。

在针对父节点为 p 的顶点 v 的每个动态规划单元 $\mathrm{DP}(v,\cdot)$ 进行构造时，我们还将确保当 x 非常小时或非常大时，不会违反边 (p,v) 的容量。更具体地说，我们将确保 $\mathrm{DP}(v,\cdot)$ 满足以下附加性质

即，对于所有 $x > \text{cap}(p, v)$，$\text{DP}(v, x) = \infty$ for $x < -\text{cap}(p, v)$，且 $\text{DP}(v, x) = \text{DP}(v, \text{cap}(p, v))$。我们将此性质称为可行容量性质。

情形 1： ***v* 是一个叶节点。** 假设 v 是一个叶节点。我们将 $\text{DP}(v, \cdot)$ 初始化为在整个 $\mathbb{R}$上取值均为 ∞ 的函数。接下来，根据 v 是否可以作为源点，至多向 $\text{DP}(v, \cdot)$ 添加两段。

首先，假设 v 可以作为源点。那么，我们可以向 v的父节点 p 发送至多 $\text{cap}(p, v)$ 的流。此外，由于 v 是一个叶节点，T_v中恰有一个源点。因此，我们更新 $\text{DP}(v, \cdot)$，并对所有 $x \geq -\text{cap}(p, v)$，令 $\text{DP}(v, x) = 1$。这会为 $\text{DP}(v, \cdot)$ 添加一段。

其次，假设 v 不是源点。那么，如果 $x \geq d(v)$ 且 $\text{cap}(p, v) \geq d(v)$，$v$ 可以从其父节点获得满足其全部需求 $d(v)$ 的流，并且该流是可行的，因为没有超过边 (p, v) 的容量。因此，如果 $\text{cap}(p, v) \geq d(v)$，我们就再次更新 $\text{DP}(v, \cdot)$，并添加满足以下条件的一段：对所有 $x \geq d(v)$，$\text{DP}(v, x) = 0$。如果 $x \geq d(v)$，但 $d(v) > \text{cap}(p, v)$，那么我们不做任何处理，因为 v 的父节点无法满足 v的需求。

注意到，$\text{DP}(v, \cdot)$ 是一个至多包含三个分段的单调递减函数。此外，它显然满足可行容量性质和定义 8 的性质 (3)。

情况 2： ***v* 不是叶节点。** 假设 v 不是叶节点，且 v 有子节点 v_1 和v_2，以及一个父节点 p。回想一下，我们假设已经为 v的两个子节点计算出了动态规划表项 $\text{DP}(v_1, \cdot)$ 和 $\text{DP}(v_2, \cdot)$。现在我们将说明如何计算两个动态规划解 $\text{DP}_A(v, \cdot)$ 和 $\text{DP}_B(v, \cdot)$，具体取决于 v 是否为源点（情况 A）或不是（情况 B）。然后，如果 v 可以用作源点，我们就设置

$$\text{DP}(v, \cdot) = \min\{\text{DP}_A(v, \cdot), \text{DP}_B(v, \cdot)\},$$

其中，我们通过引理 6 计算最小值运算。如果 v 不能用作源点，我们就设置 $\text{DP}(v, \cdot) = \text{DP}_B(v, \cdot)$。

情况 *A*：假设 v 被用作源点。我们将 $\text{DP}_A(v, \cdot)$ 初始化为在 $\mathbb{R}$的所有位置上取值为 ∞ 的函数。现在，由于 v 可以用作源点，v 可以向其父节点发送流 $\text{cap}(p, v)$，并向其子节点发送流 $\text{cap}(v, v_1)$ 和 $\text{cap}(v, v_2)$。因此，对于 $x \geq -\text{cap}(p, v)$，$\text{DP}_A(v, x)$ 中源点的数量为 1（因为 v 是源点）加上 当 v_1 可以从其父节点 v 接收流 $\text{cap}(v, v_1)$ 时，我们在 T_{v_1} 中所需的源点数量再加上 T_{v_2}对应的相同数量。因此，只需设置

$$\text{DP}_A(v, x) = 1 + \text{DP}(v_1, \text{cap}(v, v_1)) + \text{DP}(v_2, \text{cap}(v, v_2))$$

对于所有 $x \geq -$容量(p, v)。注意，这里我们利用了函数动态规划$(v_1, \cdot)$ 和动态规划$(v_2, \cdot)$ 均单调递减，且都满足可行容量性质这一点。由此可知，在这种情况下，动态规划$_A(v, \cdot)$ 是一个包含两个分段的单调递减函数。

情况 *B*：假设 v 未被用作源点。我们将动态规划$_B(v, \cdot)$ 初始化为一个在所有 $\mathbb{R}$ 上取值为 ∞ 的函数。为了计算动态规划$_B(v, x)$ 的值，我们需要求出源点数量的最小值，使得 v 从其父节点接收的流不超过 x，并且满足 T_v 中的所有需求。由于 v 不是源点，其需求 $d(v)$ 必须由其父节点 p 或其子节点（或两者共同）满足。因此，为了实现这一点，我们必须选择子节点的解动态规划(v_1, x_1) 和动态规划(v_2, x_2)，使得$d(v) \leq x - x_1 - x_2$。

由于我们没有将 v 设为源点，$\text{DP}_B(v, x)$ 中源点的数量就是我们需要放置在子树 T_{v_1} 和 T_{v_2}中的源点数量。因此，我们得到

$$\text{DP}_B(v, x) = \min_{x_1 \in \mathbb{R}}\{\text{DP}(v_1, x_1) + \text{DP}(v_2, x - x_1 - d(v))\},$$

其中我们利用了 $x_2 \le x - x_1 - d(v)$，并根据 $\mathrm{DP}(v_2, \cdot)$ 的单调性，若考虑 $x_2 = x - x_1 - d(v)$，则可以使 T_{v_2} 中的源点数量最小化。这里，我们计算出的 $\mathrm{DP}_B(v, x)$ 的流被设为可行，因为解 $\mathrm{DP}(v_i, \cdot)$ 满足可行容量性质，因此不会违反通向子节点的边约束。

由于上述等式对所有 x 的取值均成立，因此 $\mathrm{DP}_B(v,\cdot)$ 对应于两个单调递减函数的移位 (min $+$)-卷积。具体而言，根据引理 6，我们可以先计算移位后的函数 $\mathrm{DP}(v_2, \cdot - d(v))$，然后设置

$$\mathrm{DP}_B(v, \cdot) = \mathrm{DP}(v_1, \cdot) \oplus \mathrm{DP}(v_2, \cdot - d(v)),$$

我们通过引理 7 计算该值。

最后，作为后处理步骤，如果 v 可用作源点，则设置 $\mathrm{DP}(v, \cdot) = \min\{\mathrm{DP}_A(v, \cdot), \mathrm{DP}_B(v, \cdot)\}$；否则，设置 $\mathrm{DP}(v, \cdot) = \mathrm{DP}_B(v, \cdot)$，正如上文所述。但我们还需要确保 $\mathrm{DP}(v, \cdot)$ 满足可行容量性质。因此，对于 $x < -\mathrm{cap}(p, v)$，我们设置 $\mathrm{DP}(v, x) = \infty$；对于 $x > \mathrm{cap}(p, v)$，则设置 $\mathrm{DP}(v, \cdot) = \mathrm{DP}(v, \mathrm{cap}(p, v))$。注意，对 $\mathrm{DP}(v, \cdot)$ 所做的这些修改，可以在线性于 $\mathrm{DP}(v, \cdot)$ 的分段数量的时间内完成。

动态规划的性质。注意，在上述动态规划中，对于每个顶点 v，我们只需要其子节点 v_1 和 v_2 的动态规划解。因此，我们的依赖图就是输入树 T，其中所有边都指向根节点。这意味着依赖图中的每个节点只能到达通向根节点的路径上的那些节点，因此，定义 8 的性质 (2) 得以满足，其中 h 是 T 的高度。此外，可以验证，上述所有操作也满足定义 8 的性质 (3)。最后，注意在每一步中，我们只使用了引理 6 中的常数次操作，以及至多一次来自引理 7 的 (min $+$)-卷积。

E.2 近似动态规划

现在，我们说明如何通过计算近似解 $\mathrm{ADP}(v, \cdot)$，更高效地求解上述动态规划。这将揭示定义 8 中的过程 $\mathcal{P}_i$。

在我们的近似算法中，除了将每个精确解 $\mathrm{DP}(v, \cdot)$ 替换为近似解 $\mathrm{ADP}(v, \cdot)$ 之外，其他步骤都与上述完全相同。随后，我们添加一个后处理步骤，对 $\mathrm{ADP}(v, \cdot)$ 进行舍入，即令

$$\mathrm{ADP}(v, \cdot) = \lceil \mathrm{ADP}(v, \cdot) \rceil_{1+\delta} \tag{12}$$

对于参数 $\delta > 0$，我们将在稍后设定其值。

请注意，除了会造成 $\alpha = 1 + \delta$ 倍损失的舍入步骤外，我们的所有操作都是精确的。因此，定义 8 的性质 (4a) 得到满足。此外，请注意，在每一步中，我们仅使用了引理 6 中的常数次操作，以及至多一次引理 7 中的 (min $+$)-卷积。这意味着性质 (4b) 得到满足。此外，我们考虑的所有函数都是单调的，并且舍入步骤确保每个行 $\mathrm{ADP}(v, \cdot)$ 至多有 $p = O(\log_{1+\delta} n)$ 个分段。因此，性质 (4c) 也得到满足。

这意味着该动态规划是 $(h, 1+\delta, O(\log_{1+\delta}(n)))$-良好态的。应用定理 9，并取 $\delta = \ln(1+\epsilon)/(h+1)$，我们得到以下命题，它表明在二叉树上，该近似算法可以计算出一个双准则 $(1+\epsilon, 1)$-近似解。我们注意到，当 ϵ 为常数时，运行时间实质上变为 $\tilde{O}(n \cdot h^2)$，其中 h 是树的高度。因此，对于高度为 $O(1)$ 的树，我们可以得到近线性运行时间。

▶ **命题 33**。设 $\epsilon > 0$。该近似算法在时间 $O(n \cdot (h\log(n)/\epsilon)^2 \log(h\log(n)/\epsilon))$ 内，为二叉树上的同时源位置问题计算一个双准则 $(1+\epsilon, 1)$-近似解，其中 h 是树的高度。

E.3 推广到一般图（定理 4 的证明）

我们通过将问题归约到二叉情形来证明定理 4。

首先，假设 G 是一棵度数可能无界的树。接着，我们使用引理 29 证明中的相同构造，将 G 转换为二叉树 T。具体而言，我们将 G 中的每个顶点 u 替换为一棵平衡二叉树 τ_u，该树有 deg(u) 个叶节点 c_u、v_1、…、c_u、$v_{\deg(u)}$，其中 v_i 是 G 中 u 的子节点；τ_u 的内部边容量为 ∞，并将每棵 τ_u 的根节点记为 r_u。此外，对于 G 中的每条边 (u, v)，我们在 T 中插入边 (c_u,v, r_v)，其容量为 cap(c_u,v, r_v) = cap(u, v)。根据引理 29 证明中的相同论证，T 有 $O(n)$ 个顶点，高度为 $O(h\log\ n)$，其中 h 是 G 的高度。不难看出，采用这种构造时，u 到 v 在 G 中存在流，当且仅当 r_u 到 r_v 在 T 中存在流。现在，我们已经在 T 中设定了边容量，接下来只需设定顶点需求。对于 T 中的每个顶点 r_u，我们设定 $d(r_u) = d(u)$；对于 T 中所有其他顶点 v，我们设定 $d(v) = 0$。此外，在同时源位置问题的实例中，我们规定每个顶点 r_u 都可以被选为源点，而 T 中的其他顶点均不能被选为源点。注意，G 中的源点与 T 中的源点之间存在一一对应关系。结合上述关于流的观察可知，在 T 上求解同时源位置问题，即可得到 G 的一个解。

为了获得树的双准则 $(1+\epsilon, 1)$-近似结果，我们对 T 应用命题 33 中的近似算法。

最后，为了获得一般图 G 的 $(1+\epsilon,\ O(\log^4 n))$-近似解，我们按以下步骤进行。根据引理 29，我们为 G 构建二叉化Räcke树 T，并回顾一下，T 的质量为 $q = O(\log^4 n)$，高度为 $\tilde{O}(1)$。在 T 中，我们设置相应的位，表示所有叶节点都可以用作源点，而其他顶点都不能用作源点。我们对 T 应用命题 33 中的近似算法，从而在时间 $O(n)$ 内，在 T 上得到一个 $(1+\epsilon, 1)$-近似˜解。现在指出，定理 15 中的 Räcke 树（因此，引理 29 中的二叉化Räcke树也）还是一个树流稀疏器。也就是说，如果在 G 中存在一个可行流 F，那么在 T 中相应的叶节点之间就存在一个值相同的流。此外，对于 T 中叶节点之间任意一个值为 v 的可行流 F，在 G 中相应的顶点之间都存在一个值为 $\frac{1}{q}v$ 的可行流。因此，如果允许 G 中的边容量超出 $q = O(\log^4 n)$ 倍，那么我们在 T 中计算出的流在 G 中就是可行的。由此可知，我们可以在时间 $\tilde{O}(n)$ 内计算出一个 $(1+\epsilon,\ O(\log^4 n))$-近似解。

E.4 推广到动态设置（定理 5 的证明）

为了证明定理 5，我们先考虑动态二叉树这一特殊情形（定理中未提及）。我们证明，对于二叉树，可以维护一个双准则 $(1+\epsilon, 1)$-近似解，其最坏情况更新时间为 $\tilde{O}(h^3/\epsilon^2)$，其中 h 是树高度的上界。然后，我们证明定理中的结果可以由此结果推导得出。

考虑一棵动态二叉树，并在其上维护第 E 节中近似动态规划。我们将利用 T 与动态规划的依赖树相一致这一点。因此，T 中的更新会触发依赖树中的相同更新。注意，更新操作设置需求(v, d) 会导致 ADP($v,\ \cdot$) 发生变化。然后，我们可以利用定理 10 重新计算全局近似动态规划表。由于动态规划具有良好态，这棵树的高度至多为 h；又因为我们设置了 $\delta = O(h/\epsilon)$，定理表明，重新计算近似动态规划解需要 $\tilde{O}(h^3/\epsilon^2)$ 的时间。类似地，对于设置容量(($u,\ v$), c)，我们也可以再次更新

行 ADP(u, $\cdot$) 和 ADP(v, $\cdot$)，并利用定理 10 更新整个动态规划表。根据上述相同的论证，这需要 $\tilde{O}(h^3/\epsilon^2)$ 的时间。对于删除(u, v)，我们从 T 中删除边 (u, v)，根据前述相同的推理，更新时间为 $\tilde{O}(h^3/\epsilon^2)$。最后，考虑插入(($u$, v), c)，其中假设 v 成为 u 的子节点。此时可能会遇到一个问题：更新前，v 不是其连通分量的根节点。为解决这一问题，我们采用 D.5 节所述的相同重新选根过程。如 D.5 节所述，这只会重新计算 $O(h)$ 个动态规划单元的解，因此总更新时间仍为 $\tilde{O}(h^3/\epsilon^2)$。

接下来，我们证明定理 5 中的结果。首先，考虑所有边容量均设为 1，且我们希望得到一个双准则 $(1+\epsilon, n^{o(1)})$-近似解，摊销更新时间为 $n^{o(1)}/\epsilon^2$，预处理时间为 $O(n^2)$ 的情形。令 G 为动态输入图。按照引理 31，我们为 G 维护动态二叉化 Räcke 树 T，并注意到定理 16 中的动态 Räcke 树也是一种树流稀疏器。注意，对 G 的任何更新都会触发对 T 的一次更新操作，其摊销更新时间为 $n^{o(1)}$。在 T上，我们允许叶节点充当源点，但不允许其他顶点这样做。此外，我们将 T 中叶节点的需求设为 G中对应顶点的需求；其他所有顶点的需求均为 0。现在，我们使用上一段中用于二叉树的数据结构，在 T上维护一个动态双准则 $(1+\epsilon, 1)$-近似解。也就是说，当 G中的顶点需求发生变化时，我们更新 T中对应的顶点需求。当由于引理 31 中的子程序而在 T 中插入或删除一条边时，我们就更新上一段中的数据结构，该结构用于维护 T上的动态规划解。根据与 E.3 节相同的论证，由于 T 的质量为 $n^{o(1)}$，如果我们允许 G 中的边容量超出 $n^{o(1)}$ 倍，那么 T 中的任何可行流在 G 中也可行。这就推出了定理中所述的结果。

如果 G 是一棵高度为 h 但（可能）度数无界的树，我们可以维护一个双准则 $(1+\epsilon, 1)$-近似解，其最坏情况更新时间为 $\tilde{O}(h^3/\epsilon^2)$，预处理时间为 $O(n^2/\epsilon^2)$，与上述类似。也就是说，我们使用证明引理 31 时所用的相同过程，将 G 转换为一棵二叉树：把每个顶点 u 替换为一棵以 r_u为根节点的子树 τ_u。与我们在 E.3 节中的论证类似，我们只允许顶点r_u 作为 T 中的根节点，因此 T 中的任意流都对应于 G中的一个流。然后，在 T上应用用于二叉树的动态数据结构，即可得到所述结果。

最后，我们考虑这样一种情形：我们希望在只允许执行更新操作 设置需求(v, d) 时，获得双准则近似算法。在这种情况下，我们注意到底层图是静态的，因为只有顶点需求会发生变化。因此，对于输入图 G，我们可以构建定理 15 中的 Räcke 树，它同时也是树流稀疏器，并按照引理 31 将其二叉化，得到版本 T。请注意，构建这棵质量为 $\tilde{O}(\log^4 n)$ 的树需要时间 $\tilde{O}(m)$。给定这样一棵静态 Räcke 树，我们可以使用上文针对二叉树的动态数据结构，在 T 上支持 设置需求(v, d) 操作。由于 T 的高度为 $\tilde{O}(1)$，我们得到具有双准则 $(1+\epsilon, O(\log^4 n))$ 的近似结果。若要获得双准则 $(1+\epsilon, O(\log^2 n \log\log n))$ 近似，我们只需完全沿用上述方法，但将定理 15 中的 Räcke 树替换为 Harrelson、Hildrum 和 Rao [37] 的 Räcke 树；该树可用作树流稀疏器。由于其质量为 $O(\log^2 n \log\log n)$，且可在 poly(n) 时间内构建，我们便得到所需结果。

F 调整次数界

本节讨论我们推导出的重配置次数界。为说明这些下界的意义，请注意，“经典”动态算法会在内存中维护一个显式解，并具有多对数更新时间；这在许多实际场景中都很理想。然而，一些动态算法（例如上文的动态规划算法）只能在多对数时间内返回近似解的值，这有时称为隐式解。为了了解对于我们研究的问题而言，隐式解是否必不可少，我们考虑这样一类算法：它们维护多个显式解，其中只需有一个解可行。我们认为这是一个值得研究的有趣设定，因为它本质上介于上述两种情形之间。如果即使是维护多个解的算法也必须付出很高的重配置代价，这就表明隐式解或许不可避免。下面我们将证明，对于全动态背包问题和全动态 k-平衡划分，情况正是如此。

这里，我们考虑输入通过更新操作进行插入和删除的动态算法。这些算法可以维护多个显式解，并且必须确保每个时间步之后都存在一个满足特定保证的解，同时尽量减少更新这些解所需的重配置次数。更具体地说，我们考虑在每个时间步显式维护 s 个解 $S_1^{(t)}$，……，$S_s^{(t)}$的算法。这里，我们假设每个时间步之后都会执行一次更新操作，之后算法可以修改其解。我们称算法维护了一个 α-近似解，如果对于每个时间步 t，都存在一个索引 $i = i(t)$，使得 $S_i^{(t)}$ 是我们所研究问题的一个可行且 α-近似解。

请注意，在这种设定下，算法所需的重配置次数可能会少得多，因为在每个时刻 t，它都可以选择不同的解。因此，在更新操作之后，它可能不必对任何一个解进行大幅更新。还要注意，这种要求每个时间步都存在一个可行解的概念，在某种程度上让人联想到编码理论中的列表译码：译码器可以输出一组消息，只需确保正确的消息包含在其中即可。

重配置的衡量方式取决于具体问题，具体而言，取决于该问题的解如何存储。一般来说，给定连续两个时间步的两个解，我们令$d(S_i^{(t)}, S_i^{(t+1)})$ 表示时间步 t 中第 i 个解所产生的（与问题相关的）重配置（关于如何为我们研究的问题设定 $d(\cdot,\cdot)$，见下文）。算法的总重配置为

$$\sum_t \sum_{i=1}^{s} d(S_i^{(t)}, S_i^{(t+1)}).$$

接下来，我们将介绍推导出的具体重构次数下界。

背包问题的重配置界。 在背包问题中，解对应的就是背包中所包含的物品子集。为了衡量重配置，我们设定$d(S_i^{(t)}, S_i^{(t+1)}) = \left|S_i^{(t)} \triangle S_i^{(t+1)}\right|$，即考察时间步 t 和 $t+1$ 中第 i 个解的对称差的基数。

我们的主要结果表明，对于固定精度 ϵ，任何动态 $(1-\epsilon)$-近似算法都必须维护 $\Omega(1/\epsilon)$ 个解，或者其重配置必须达到 $\Omega(\binom{n}{\epsilon})$，即使每次只插入一个物品也是如此。

▶ **定理 34。** 设 $\epsilon \in (0, 1/2)$。假设 $s < 8\epsilon^{1(1+2\epsilon)}$ 且 $n \in \mathbb{N}$ 是 s的一个充分大的倍数。那么，任何用于背包问题、具有 s 个解的动态随机化$(1-\epsilon)$-近似算法，其重配置次数都必须为$\Omega(\binom{n}{s})$。即使是单项插入，这一结论也成立。

针对 k-平衡划分的重配置次数界。在 k-平衡划分中，每个解 $S_i^{(t)}$ 由 k 个簇 $V_{i,t}1, \ldots, V_{i,t}k$ 组成，这些簇将顶点集合 V 划分开来。为衡量重配置次数，我们设定 $d(S_i^{(t)}, S_i^{(t+1)}) = \sum_{j=1}^{k} \left| V_{i,t}j \triangle V_{i\,t+1}j \right|$，即统计从时间 t 到 $t+1$，其所属集合 $V_{i,\cdot}j$ 发生变化的顶点总数。

我们的主要结果表明，对于任意 C 和固定的 ϵ，任何维护 $(C, 1+\epsilon)$-近似解的算法，都必须使用 $\Omega(1/\epsilon)$ 个解，或者具有 摊销 重配置量 $\Omega(\epsilon^2 \frac{n}{k})$；即使只插入 $O(1/\epsilon)$ 条边也是如此。这里，摊销重配置量指总重配置量除以更新操作的总次数。

▶ **定理 35.** 设 $C > 0$ 为任意值，且 $\epsilon \in (0, 1/2)$。假设 $k \geq 4$ 和 $s < \frac{1}{4\epsilon}$成立。那么，对于 k-平衡划分，任何具有s 个解的动态随机化$(C, 1+\epsilon)$-近似算法，其摊销重配置量必须为$\Omega(\epsilon^2 \frac{n}{k})$。即使仅进行 $O(1/\epsilon)$ 次边插入，这一结论仍然成立。

F.1 定理 34 的证明

我们证明定理 34。我们运用 Yao 原理 [63], ，即考虑一种确定性算法，并给出一个输入分布，证明该算法的重配置次数的期望为 $\Omega(\frac{n}{s})$。

考虑一个实例，初始时有 n 个物品，每个物品 i 的权重为 $w_i = 1$，价格为 $p_i = 1$。我们将这些物品称为小物品。我们将背包问题的预算设为 $B = n$。注意，在这个实例中，OPT 为 $= n$，因为所有小物品都能装入背包。

现在，我们从 $[2s-1] = \{0, \ldots\ldots, 2s-1\}$ 中均匀随机地抽取一个整数 j，并设定 $k = i \cdot \frac{n}{2s} + \frac{n}{4s}$—我们插入一个重物品，其 $p = n - k + 2\epsilon n$ 且 $w = n - k$。注意，插入重物品后，OPT 为 $= n + 2\epsilon n$，因为最优解由该重物品和 k 个小物品组成。

我们用 S_1、……、S_s 表示插入重物品之前算法维护的解，用 S'_1、……、S'_s 表示插入重物品之后的解。我们用 small(S_i) 表示解 S_i 中小物品的数量。

接下来，我们将证明，任何一个 $(1-\epsilon)$-近似解 S'_i 都必须包含重物品和“几乎” k 个小物品。然而，我们还将证明，以常数概率，所有解 S_i 最初包含的小物品数量都“远少于”或“远多于” k。由此可知，获得任何一个 $(1-\epsilon)$-近似解都必须付出高昂的重配置代价。

我们按相反的顺序采用这一证明策略。首先证明，以常数概率，所有 S_i 包含的小物品数量都“远少于”或“远多于” k。

▶ **引理 36。** 以至少 $1/2$ 的概率，对所有 $i \in [s]$，都有 $|k - \mathit{small}(S_i)| \geq \frac{n}{4s}$。

证明。假设我们将集合 $\{1, \ldots, n\}$ 划分为 $2s$ 个连续区间，每个区间长度为 $\frac{n}{2s}$。注意，k 是其中一个区间的中点。此外，由于只有 s 个解，而有 $2s$ 个区间，因此至少有一半的区间不包含集合 $\{\text{small}(S_i)\colon i = 1, \ldots, s\}$ 中的数；我们称这些区间为空区间。因此，以至少 $1/2$ 的概率，k 是一个空区间的中点。如果包含 k 的区间为空，那么对于所有 $i = 1, \ldots, s$，k 到 small(S_i) 的距离至少为 $\frac{n}{4s}$。◀

接下来，回想一下，$S'_1, \ldots, S'_n$ 是算法在插入重物品后维护的解。我们证明，任何 $(1-\epsilon)$-近似解都必须包含该重物品和一些小物品。

▶ **引理 37。** 假设 S'_i 是一个$(1-\epsilon)$-近似解。那么 S'_i 包含重物品和正数个小物品。

证明。 首先，假设一个解 S_i' 只包含小物品。那么其总价格至多为 n。然而，我们有

$$(1-\epsilon)\,\mathrm{OPT} = (1-\epsilon)(1+2\epsilon)n > n,$$

其中我们用到了 $\epsilon < 1/2$。因此， S_i' 不是一个 $(1-\epsilon)$-近似解。

其次，假设 S_i' 只包含重物品。那么我们有

$$\begin{aligned}(1-\epsilon)\,\mathrm{OPT} &= (1-\epsilon)(p+k)\\ &= p+k-\epsilon(p+k)\\ &\geq p+\frac{n}{4s}-\epsilon(1+2\epsilon)n\\ &> p+2\epsilon(1+2\epsilon)n-\epsilon(1+2\epsilon)n\\ &> p,\end{aligned}$$

其中我们用到了 $k \geq \frac{n}{4s}$，$p+k = n+2\epsilon n$ 和 $s < \frac{1}{8\epsilon(1+2\epsilon)}$。因此，仅包含重物品的解不是 $(1-\epsilon)$-近似解。◀

接下来，我们证明任何 $(1-\epsilon)$-近似解都必须包含“几乎” k 个小物品。

▶ **引理 38.** 假设 S_i' 是一个$(1-\epsilon)$-近似解。那么

$$k - \frac{n}{8s} \leq \mathit{small}(S_i') \leq k.$$

证明。 上界源于以下事实： S_i' 必须包含权重为 $n-k$ 的重物品（根据引理 37），之后只能纳入 k 小物品，因为预算约束设定为 $B = n$。

为证明下界，注意到由于 S_i' 是一个 $(1-\epsilon)$-近似解，我们有其解的值为

$$p + \mathsf{small}(S_i') \geq (1-\epsilon)\,\mathrm{OPT} = (1-\epsilon)(p+k).$$

因此，我们得到

$$\begin{aligned}\mathsf{small}(S_i') &\geq k-\epsilon(p+k)\\ &= k-\epsilon(1+2\epsilon)n\\ &> k-\frac{n}{8s},\end{aligned}$$

这里用到了 $s < \frac{1}{8\epsilon(1+2\epsilon)}$。◀

为完成定理的证明，我们以引理 36 中的事件为条件，即对于所有 $i \in [s]$，都有 $|k - \mathsf{small}(S_i)| \geq \frac{n}{4s}$。

现在考虑插入重物品后的任意解 S_i'。如果 S_i' 不是 $(1-\epsilon)$-近似解，我们可以忽略 S_i'。如果 S_i' 是 $(1-\epsilon)$-近似解，那么根据引理 38，它满足 $k - \frac{n}{8s} \leq \mathsf{small}(S_i') \leq k$。由于我们假设引理 36 中的事件成立，算法必须向 S_i 中至少插入/从中删除 $\frac{n}{8s}$ 个小物品，才能得到 S_i'。

由于引理 36 中的事件以至少 $1/2$ 的概率发生，且上述论证对所有 $(1-\epsilon)$-近似解 S_i' 均成立，因此期望调整量为 $\Omega(\frac{n}{s})$。

F.2 定理 35 的证明

我们证明定理 35。再次应用 Yao 原理 [63]，即考虑一个确定性算法，并给出一个输入分布，证明该算法的摊销重分配量在期望意义下为 $\Omega(\epsilon^2 \frac{n}{k})$。

我们考虑一个包含 n 个顶点的图。初始实例由 $\frac{k}{2\epsilon}$ 个星形图组成，每个星形图包含 $2\epsilon\frac{n}{k}$ 个顶点。注意，在这里，一个最优解会将恰好来自 $\frac{1}{2\epsilon}$ 个星形图的顶点放入每个划分 V_j 中；不同 V_j 中的顶点之间没有边，因此最优割值为零。因此，任何 $(C,\ 1+\epsilon)$-近似解的解也必须具有零割值。

在更新阶段，我们从星形图的中心节点之间均匀随机地采样 s 条边，并将其插入图中。注意，插入这些边后，我们至多连接了 s 个星形图，且最大的连通分量的大小至多为 $s \cdot 2\epsilon\frac{n}{k} \leq \frac{1}{2}\frac{n}{k}$，这里我们利用了 $s \leq \frac{1}{4\epsilon}$。因此，最优解仍然具有零割值，故任何 $(C,\ 1+\epsilon)$-近似解都必须具有零割值。

接下来，我们分析一个从初始解$S_1^{(0)}, \ldots, S_s^{(0)}$开始的算法的重配置。首先，我们证明，在时刻 0 将某个星形图拆分得“过于彻底”的解必然导致很高的重配置。其次，我们考察所有其他解，并证明我们的插入操作仍会在期望意义下引发很高的重配置。

如果解 $S_i^{(0)} = \{V_{(i,0)1}, \ldots, V_{(i,0)k}\}$ 对所有星形图 H 都满足：存在一个索引 j，使得 $V_{(i,0)j}$ 包含来自 H 的至少 $\epsilon\frac{n}{k}$ 个顶点，且至多有 $\epsilon\frac{n}{k}$ 个顶点被放置在 $\bigcup_{j' \neq j} V_{(i,0)j'}$ 中，那么我们称该解为有用的。给定一个星形图 H 和一个解 $S_i^{(0)}$，我们用 $j(H, i)$ 表示满足 $V_{(i,0)j}$ 包含来自 H 的至少至少 $\epsilon\frac{n}{k}$ 个顶点的索引 j。如果一个解不是有用的，我们就称其为无用的。

首先，考虑无用的解。存在两种情况。情况 A：假设存在一个星形图 H，使得对于所有索引 j，都有 $\bigcup_{j' \neq j} V_{j'}^{(i,0)}$ 中包含来自 H 的顶点数多于 $\epsilon\frac{n}{k}$。注意，如果算法想在边插入完成后使用这个解，就必须确保割值为零。因此，它必须将至少 $\epsilon\frac{n}{k}$ 个顶点移动到某个 $V_{(i,0)j}$ 中，这需要进行 $\left|\bigcup_{j' \neq j} V_{j'}^{(i,0)}\right| \geq \epsilon\frac{n}{k}$ 次顶点移动。情况 B：假设存在一个星形图 H，使得对于所有 j，都有 $V_{(i,0)j}$ 中来自 H 的顶点数少于 $\epsilon\frac{n}{k}$。根据 $\epsilon \in (0, 1/2)$，在这种情况下，算法也必须移动至少 $(1-\epsilon)$ 个 $\frac{n}{k} \geq \epsilon\frac{n}{k}$ 顶点，以便在更新完成时，所有 H 最终都包含在同一个集合 $V_{(i,s)j}$ 中。由此可知，对于无用的解，在对 $s \leq \frac{1}{\epsilon}$ 次插入操作进行摊还分析后，我们的定理仍然成立。

其次，在证明的剩余部分中，只考虑有用的解 $S_i^{(0)} = \{V_{(i,0)1}, \ldots, V_{(i,0)k}\}$。注意，当我们在两个星形图 H_1 和 H_2 之间插入一条边时，如果 $j(H_1, i) \neq j(H_2, i)$，那么算法必须移动至少 $\epsilon\frac{n}{k}$ 个顶点，以确保在 s 次插入操作完成后，来自 H_1 和 H_2 的所有顶点都被放在同一个集合 $V_j^{(i,s)}$ 中，其中 j 取某个值。我们称对于解 i 而言，这种插入操作是昂贵的。

注意，如果我们的边插入操作中包含一种对所有解都适用的昂贵插入，那么更新任意一个满足 $S_i^{(s)}$ 为 $(C, 1+\epsilon)$-近似的解 $S_i^{(0)}$，至少会产生 $\epsilon\frac{n}{k}$ 的重配置。证明的其余部分将用于说明，这一事件以常数概率发生。这就证明了该定理。

我们先考虑一个固定的解 $S_i^{(0)}$，以及在随机选取的星形图 H_1 和 H_2 之间进行一次随机边插入。回顾一下，总共有 $\frac{k}{2\epsilon}$ 个星形图。此外，对于所有 j，我们有 $\left|V_{(j,i0)}\right| \leq (1+\epsilon)\frac{n}{k}$，因此对于每个 j，至多有 $\frac{(1+\epsilon)\epsilon}{\ }$ 个星形图 H 满足 $j = j(H, i)$。因此，边插入

代价高昂的概率为

$$\begin{aligned}\mathbf{Pr}\left(j(H_1,i)\neq j(H_2,i)\right) &= 1-\mathbf{Pr}\left(j(H_1,i)=j(H_2,i)\right)\\ &\geq 1-\frac{(1+\epsilon)/\epsilon}{k/(2\epsilon)}\\ &= 1-\frac{2(1+\epsilon)}{k}\\ &\geq \frac{1}{2},\end{aligned}$$

这里我们用到了 $k\geq 4$。

接下来，我们考虑一个固定解 $S_i^{(0)}$，以及在星形图之间独立且均匀随机选取的 s 次边插入。于是，以至少 $1-2^{-s}$ 的概率，这些边插入中至少有一次对于解 i 来说代价高昂。

最后，注意到对于所有 解 i，存在一次代价高昂的边插入的概率至少为

$$\begin{aligned}\left(1-2^{-s}\right)^s &= \exp\left(s\ln(1-2^{-s})\right)\\ &\geq 1+s\ln(1-2^{-s})\\ &\geq 1-s2^{-s}\\ &\geq \frac{1}{4},\end{aligned}$$

这里我们用到了，对于所有 $x\in\mathbb{R}$，$\exp(x)\geq 1+x$；当 x 接近 1 时，$\ln(x)$ 的泰勒展开；以及对于所有 s，$s2^{-s}\leq\frac{3}{4}$ 这一事实。

我们得出结论：以常数概率，对于所有解 i，都存在一次代价高昂的边插入。在这种情况下，该算法的总重配置至少为 $\epsilon\binom{n}{k}$。因此，该算法的期望总重配置为 $\Omega(\epsilon\binom{n}{k})$。由于我们只执行了 s 次边插入，这意味着摊还重配置为 $\Omega(\epsilon^2\binom{n}{k})$。

G 非单调函数与 ℓ_∞-项链对齐

到目前为止，我们只考虑了单调分段常值函数。现在，我们将把一些结果推广到具有多个非单调性的分段常值函数，并在 G.1 节中给出详细内容。我们还将在 G.2 节中推导出用于 ℓ_∞-项链问题的新近似算法。特别是，对于 ℓ_∞-项链问题，我们提出了首个运行时间近线性且加性误差为 ϵ 的近似算法。我们还提出了该问题的首个动态近似算法，其加性误差为 ϵ，更新时间为 $O((1/\epsilon)^2\log(1/\epsilon))$；该算法在以空向量 x 和 y 为初始输入时，预处理时间为 $O(1)$，并且只需次线性空间 $O(1/\epsilon)$。有关结果的详细内容，请参见定理 44。

G.1 具有非单调性的分段常值函数

现在我们将说明，即使分段常值函数包含非单调性，我们仍能对其执行高效操作。不过，我们的子程序的运行时间通常会受到函数中非单调性数量的影响。

下面我们将非单调性形式化。若存在若干值 $f\colon [0,t)\to[0,W]\cup\{-\infty,\infty\}$，使得 k 单调区段 成立，则称函数

使得在每个区间 $[x_i, x_{i+1})$ 上， f 都是单调的。这里要求 f 在所有 区段上单调递减，或者在所有 区段上单调递增。注意，单调函数有 $k = 1$ 个单调区段（令 $x_0 = 0$ 和 $x_1 = t$ 即可），并且点 x_1，……， x_{k-1} 可视为 f 非单调的点。

一个关键操作仍将是舍入。不过，与之前不同，我们主要讨论将数舍入到 δ 的倍数，而不是舍入到 $1 + \delta$ 的幂。这将便于我们之后将结果应用于 ℓ_∞-项链。我们还将简要说明如何将本小节的结果扩展到舍入到 $1 + \delta$ 的幂的情形。

接下来，令 $\delta > 0$，并考虑一个简单的舍入函数，将数向下舍入到 δ 的倍数。更具体地，对于 $y \in \mathbb{R}$，我们定义 $\mathsf{b}y\mathsf{c}^*_\delta = \max\{i \cdot \delta \colon i \cdot \delta \le y, i \in \mathbb{Z}\}$，并约定 $\mathsf{b}-\infty\mathsf{c}^*_\delta = -\infty$ 以及 $\mathsf{b}\infty\mathsf{c}^*_\delta = \infty$。我们还将舍入操作扩展到函数 $f \colon [0,t) \to [0,W] \cup \{-\infty, \infty\}$，定义 $\mathsf{b}f\mathsf{c}^*_\delta \colon [0,t) \to [0,W] \cup \{-\infty, \infty\}$ 为满足以下条件的函数：对于所有 $x \in [0,t)$，有 $\mathsf{b}f\mathsf{c}^*_\delta(x) = \mathsf{b}f(x)\mathsf{c}^*_\delta$。接下来，我们证明函数 $\mathsf{b}f\mathsf{c}^*_\delta$ 可以高效计算，并且仅包含少量分段。

▶ **引理 39.** 设 $\delta > 0$，并且 $f \colon [0,t) \to [0,W] \cup \{-\infty, \infty\}$ 是一个分段常数函数，具有 p 个分段和 k 个单调区段。那么，我们可以在 $O(p \log p)$ 时间内计算函数 $\mathsf{b}f\mathsf{c}^*_\delta$，且 $\mathsf{b}f\mathsf{c}^*_\delta$ 有 $O(k \cdot W/\delta)$ 个分段。

证明。令 $(x_1, y_1) \ldots, (x_p, y_p)$ 表示 f 的列表表示。我们构造 $\mathsf{b}f\mathsf{c}^*_\delta$ 的列表表示 $(x'_1, y'_1) \ldots, (x'_p, y'_p)$。对于所有 $i = 1, \ldots, p$，令 $x'_i = x_i$，并令 $y'_i = \mathsf{b}y_i\mathsf{c}^*_\delta$。之后，我们合并所有具有相同 y'_i值的连续分段；这可以完全按照引理 6 的证明中所述的剪枝步骤进行。由于 f 的取值范围是$[0,W] \cup \{-\infty, \infty\}$，$[0,W]$ 中 δ 的倍数共有 $O(W/\delta)$ 种选择。特别地，在 f 的每个单调区段上，$\mathsf{b}f\mathsf{c}^*_\delta$ 有 $O(W/\delta)$ 个分段。由于 f 有 k 个单调区段，这意味着 $\mathsf{b}f\mathsf{c}^*_\delta$ 总共有 $O(k \cdot W/\delta)$ 个分段。请注意，上述所有操作都可以在线性时间内完成，而运行时间上界源于我们还需要将这些分段存储在二叉搜索树中。◀

接下来，我们证明可以在关于两个分段常值函数的分段数呈二次关系的时间内，计算它们的 $(\min, +)$-卷积。该引理推广了引理 7 的结果，因为我们去掉了其中一个函数必须单调的假设（但代价是证明更加复杂）。我们在 G.1.1 节中证明该引理。

▶ **引理 40。** 设 $f_1, f_2 \colon [0,t) \to [0,W] \cup \{-\infty, \infty\}$ 是分段常值函数，且各自至多有 p 个分段。那么，我们可以在 $O(p^2 \log p)$ 时间内计算 $f = f_1 \oplus f_2$，并且 f 有 $O(p^2)$ 个分段。

结合上述两个引理，我们可以证明，即使存在非单调性，也能高效计算 $(\min, +)$-卷积的加性近似。更具体地说，如果对于所有 $x \in [0,t)$，都有 $g(x) - \epsilon \le f(x) \le g(x)$，那么我们称 $f \colon [0,t) \to [0,W] \cup \{-\infty, \infty\}$ 是 $g \colon [0,t) \to [0,W] \cup \{-\infty, \infty\}$ 的加性 ϵ-近似 。现在我们得到以下定理。

▶ 定理 *41.* 设 $f, g \colon [0,t) \to [0,W] \cup \{-\infty, \infty\}$ 是两个具有 k 个单调区段的函数，并假设我们已经计算出 $\mathsf{b}f\mathsf{c}^*_\delta$和 $\mathsf{b}g\mathsf{c}^*_\delta$。那么，函数 $(\mathsf{b}f\mathsf{c}^*_\delta) \oplus (\mathsf{b}g\mathsf{c}^*_\delta)$ 是 $f \oplus g$ 的加法 2δ-近似，至多有 $O((k \cdot W/\delta)^2)$ 个分段，并且可以在 $O((k \cdot W/\delta)^2 \log((k \cdot W/\delta)^2))$ 时间内计算出来。

证明。 近似比由三角不等式得出。关于分段数量和运行时间的结论，则可由引理 39 和引理 40 结合得出。◀

我们注意到，通过将引理 39 表述为针对舍入操作 $\lceil\cdot\rceil_{1+\delta}$ （将数值舍入到 $1+\delta$ 的幂次；参见引理 6），我们可以得到定理 41 的以下版本。

▶ 定理*42.* 设 $f, g\colon [0,t) \to [0,W]\cup\{-\infty,\infty\}$ 是两个具有 k 单调区段的函数，并假设我们已经计算出 $dfe_{1+\delta}$ 和 $dge_{1+\delta}$。那么 $(dfe_{1+\delta})\oplus(dge_{1+\delta})$ 是 $f\oplus g$ 的 $(1+\delta)$-近似，最多有 $O((k\cdot log_{1+\delta}(W))^2)$ 个分段，且可在 $O((k\cdot log_{1+\delta}(W))^2\, log((k\cdot log_{1+\delta}(W))^2))$ 时间内计算出来。

这一结果推广了我们之前的方法：先通过引理 6 对单调函数进行舍入，然后应用引理 7 中高效的卷积。更具体地说，注意到单调函数只有一个单调区段，因此，对两个函数进行舍入后，我们的引理 7 算法可在时间$O(\log^2_{1+\delta}(W)\log\log_{1+\delta}(W))$ 内计算 $(\min,+)$-卷积，这与结合上述两个引理得到的运行时间相同。因此，定理42中的算法在 $k=1$ 情况下与这一结果相符，而在应用于 $k>1$ 时则对其进行了推广。

G.1.1 引理 40 的证明

我们假设 f_i 对于 $i=1,\ 2$ 以双向链表 $(x_1^i,\ y_1^i)\ldots,\ (x_p^i,\ y_p^i)$ 的形式给出，使得对于所有 $1\le j<p,\ x_j^i<x_{j+1}^i$。我们将以相同的表示形式输出 f。

为了计算 f，我们将使用以下不重叠区间数据结构 *(NOI)*。设 $[a,b]$ 和 $[a',b']$ 是实数集的两个子集。我们将其中每个称为一个区间，并称它们在 $[a,b]\cap[a',b']\neq\emptyset$ 时重叠。如果 $a\ge b$，则称区间 $[a,b]$ 为空。NOI 数据结构存储一组 S，其中包含两两不重叠且非空的区间 $I=[a,b]$，并支持以下操作：

- *ClosestLargerInterval(z)*，给定一个数 z，返回区间 $[a,b]$ 以及一个布尔值 *bool*。如果 *bool* 为真，则 $z\le b$，且 S 中不存在满足 $z\le b'<b$ 的区间 $[a',b']$。注意，z 可能属于 $[a,b]$。如果 *bool* 为假，则不存在满足 $z\le b$ 的区间 $[a,b]$，并且 a 和 b 的返回值未定义。*InsertInterval(a, b)*，将区间 $[a,b]$ 插入 S，并将其与所有和它重叠的区间合并，同时相应地更新 S。如下一条结论所述，这种数据结构存在一种高效实现，我们将在本节末尾证明这一点。

▷ 命题 43。存在一种不重叠区间数据结构的实现，使得任意包含 q 个操作的序列所需时间为 $O(q\log q)$。

我们按如下方式计算 f。注意，对于任意 $1\le s\le p$ 和 $1\le t\le p$，f_1 的函数值和 f_2 的函数值在每个二维矩形上均为常数；这些二维矩形的顶点为 $(x_s^1,\ x_t^2)$、$(x_s^1,\ x_{t+1}^2)$、$(x_{s+1}^1,\ x_t^2)$ 和 $(x_{s+1}^1,\ x_{t+1}^2)$。我们将该矩形称为 R_{st}，并用 $[x_s^1+x_t^2, x_{s+1}^1+x_{t+1}^2]$ 表示该矩形的值域 R_{st}，用 $y_s^1+y_t^2$ 表示该矩形的函数值，其中假设 $\infty+y$ 在 $y\in W_\infty$ 等于 ∞。共有 K^2 个这样的矩形。

现在注意，对于任意满足 $x_s^1+x_t^2\le x\le x_{s+1}^1+x_{t+1}^2$ 的值 x，也就是说，x 位于矩形 R_{st} 的值域内，函数值 $y_s^1+y_t^2$ 是计算 $f(x)=\min_{\bar{x}}\{f_1(\bar{x})+f_2(x-\bar{x})\}$ 时出现的各个和之一。我们将通过比较所有值域包含 x 的矩形 R_{st} 的函数值，来计算 $f(x)$（对所有值 x“同时”计算）。我们所利用的关键观察如下：由于我们将按函数值递减的顺序考察这些矩形，因此，按此顺序，第一个值域包含 x 的矩形，其函数值就等于 $f(x)$。

因此，处理一个矩形时，我们需要确定所有尚未被赋予函数值的值域，也就是 $[0,t]$ 的子区间。为此，我们使用 NOI 数据结构存储所有已经被赋予函数值的值 x 所对应的区间。此外，我们使用一棵平衡二叉搜索树 $\mathcal{B}$，在其叶节点中存储每个已经被赋予函数值的区间及其（常数）函数值。具体而言，我们将这些值域按较小边界值 x' 的顺序存储在 $\mathcal{B}$ 的叶节点中。两者的区别在于，NOI 数据结构会合并重叠区间，而不考虑其函数值；而存储为 B 的叶节点的每个区间都具有相同的函数值，即其 f 值为常数。

具体来说，我们按如下方式进行：首先，通过遍历 f_1 和 f_2 的列表生成所有矩形 R_{st}，并按其函数值非递减的顺序对它们进行排序。这需要 $O(p^2 \log p)$时间。然后按此顺序处理这些矩形。为此，我们首先初始化一个空的 NOI 数据结构以及一棵空的平衡二叉搜索树$\mathcal{B}$。接下来，我们描述如何处理这些矩形。设 R_{st} 是下一个待处理的矩形。我们对 R_{st}执行以下步骤：

1. $z = x_s^1 + x_t^2$2. $(a, b, bool)$ = ClosestLargerInterval(z) 3. 当 $bool$ 为真且 $b < x_{s+1}^1 + x_{t+1}^2$ 时，执行 a. 如果 $z \notin [a,b]$，则将区间 $[z,a]$ 及其函数值 R_{st} 插入 B。b. $z = b$c. $(a, b, bool)$ = ClosestLargerInterval(z)

4. 如果 $bool$ 为真，则将区间 $[z,a]$ 连同函数值 R_{st} 插入到 $\mathcal{B}$中；否则，将区间 $[z, x_{s+1}^1 + x_{t+1}^2]$ 连同函数值 R_{st} 插入到 $\mathcal{B}$中。5. $InsertInterval(x_s^1 + x_t^2, x_{s+1}^1 + x_{t+1}^2)$。

处理完所有矩形后，我们按顺序遍历$\mathcal{B}$ 的叶节点，并用双向链表将它们连接起来，从而创建函数 f 的（有序）列表表示。由于我们按函数值递增的顺序处理矩形，因此可以保证，对于每个值 x，任意矩形 R_{st} 中最小的函数值都会作为 $f(x)$ 返回。

请注意，每次向$\mathcal{B}$ 中插入所需时间为 $O(\log p)$，而对 NOI 数据结构的调用次数与矩形数量加上 NOI 数据结构中合并的区间数量成正比。处理一个矩形最多会创建一个新区间，并且合并后的区间不会再次分离，因此区间合并次数至多等于矩形数量。因此，至多有 p 次区间合并，以及至多 $2p^2$ 次向$\mathcal{B}$中的插入操作。因此，上述算法的总运行时间为 $O(p^2 \log p)$，再加上 NOI 数据结构所需的时间；根据命题 43，该时间也为 $O(p^2 \log p)$，因为 $q = O(p^2)$。

我们仍需证明命题 43。

命题 43 的证明。 我们使用平衡二叉搜索树实现 NOI 数据结构。叶节点中存储按上端点排序的不重叠区间。

该 $ClosestLargerInterval(z)$ 操作会搜索区间 $[a,b]$，使得 b 是上端点不小于 z的区间中最小的上端点。如果不存在这样的区间，则将 $bool$ 设为 false；否则将其设为 true，并返回作为区间的 $[a,b]$。注意，查找 $[a,b]$需要 $O(\log q)$ 时间，因为 q 是平衡二叉树中存储的区间的最大数量。

$InsertInterval(a,b)$ 操作首先执行一个 $ClosestLargerInterval(a)$ 操作。令 $(a', b', bool)$ 为结果。如果 $bool$ 为假，则将区间 $[a,b]$ 作为新区间插入，然后过程终止。否则，区间 $[a', b']$ 是满足 $a \leq b'$的、上端点最小的区间。注意， $[a', b']$ 可能与 $[a,b]$ 重叠，接下来我们会对此进行检查。如果 $b < a'$，则将一个值域为 $[a,b]$ 的叶节点插入平衡搜索树，并

$InsertInterval(a,b)$ 终止。否则，($b \geq a'$)，令 L 为平衡搜索树中存储 $[a',b']$ 的叶节点。如果 $b \leq b'$，则通过更新 L 以存储区间 $[\min(a,a')\ b']$ 来合并这两个区间，并且 $InsertInterval(a,b)$ 终止。但是，如果 $b > b'$，新区间 $[a,b]$ 可能与 S 中更多的区间重叠。因此，我们执行以下步骤：

1. $z = b'$2. $(a'',b'',bool)$ = ClosestLargerInterval(z) 3. 当 $bool$ 为真时，执行以下操作：a. 如果 $b < a''$，则更新叶节点 L，使其存储区间 $[\min(a,a')\ b]$，然后 InsertInterval 终止。b. 从平衡搜索树中删除存储区间 $[a'',b'']$ 的叶节点。c. 如果 $b \leq b''$，则更新叶节点 L，使其存储区间 $[\min(a,a')\ b'']$，然后 InsertInterval 终止。d. 否则，$z = b''$ 并执行 $(a'',b'',bool)$ = ClosestLargerInterval(z)。4. 更新 L，使其存储区间 $[\min(a,a')\ b]$。

请注意，该算法会将所有与 $[a,b]$ 重叠的区间合并为一个区间，并相应地更新平衡搜索树。

设 t 为 $InsertInterval(x,y)$执行的迭代次数。运行时间为$O((t+1)\log q)$，因为每次迭代都会调用一次 $ClosestLargerInterval$，在平衡二叉树中删除一个叶节点，并至多修改一个叶节点的标签。每次迭代都会使平衡二叉树中的叶节点数量减少 1。此外，每次调用 $InsertInterval$ 若未执行上述 while 循环中的任何迭代，叶节点数量最多增加 1；且没有其他操作会改变叶节点数量。由于 $InsertInterval$至多被调用 q 次，因此所有 ***InsertInterval***调用中，while 循环至多执行 q 次，每次耗时 $O(\log q)$。因此，q 次 $InsertInterval$ 调用的总运行时间为 $O(q \log q)$。◀

G.2 ℓ_∞-项链对齐

利用上述技术，我们提出了一种针对ℓ_∞-项链对齐问题 $[14,58]$的新型近似算法。该问题的输入由两条项链组成，分别表示为两个包含 n 个实数的已排序向量：$x = \langle x_0, x_1, \ldots, x_{n-1} \rangle$ 和$y = \langle y_0, y_1, \ldots, y_{n-1} \rangle$，其中 $x_i, y_i \in [0,1)$表示单位周长圆上的点。我们有时将元素 x_i 和 y_j 称为珠子。

我们将珠子 x_i 和 y_j 之间的距离定义为沿周长为单位长度的圆形项链的圆周顺时针距离和逆时针距离中的最小值，即设

$$d^\circ(x_i, y_j) = \min\{|x_i - y_j|, 1 - |x_i - y_j|\}.$$

在 ℓ_∞-项链对齐问题中，我们需要找到一个偏移量 $c \in [0,1)$和一个位移$s \in [n+1]$，使其最小化

$$\max_{i=0}^{n-1}(d^\circ((x_i + c) \mod 1, y_{(i+s) \mod n})).$$

在上述定义中，偏移量 c 表示第一条项链相对于第二条项链顺时针旋转的角度。此外，位移 s 定义了珠子之间的完美匹配，使得第一条项链上的珠子 i 与第二条项链上的珠子 $(i+s) \bmod n$ 相匹配。

Bremner 等人 [14] 证明，ℓ_∞-项链对齐问题可以在 $\tilde{O}(n^2)$ 时间内精确地求解。作为补充，我们证明可以在 $\tilde{O}(n+\epsilon^{-2})$ 时间内计算出具有加性误差 ϵ 的解。

我们还考虑问题的动态版本，其中会插入和删除珠子。更具体地说，假设初始时 x 和 y 均为空，并提供以下更新操作：

- 插入(i, α, β)，将 $\alpha \in [0,1)$ 插入 x 的第 i 个位置，并进一步将$\beta \in [0,1)$ 插入 y 的第 i 个位置。我们要求插入后，x 和 y 仍保持有序。删除(i)，从 x 中删除 x_i，并从 y中删除 y_i。

请注意，这两种操作都会改变 x 和 y 中的条目数量，但能确保 x 和 y 始终具有相同的长度。我们证明，可以在更新时间为 $O(1/\epsilon^2\log(1/\epsilon))$ 的情况下，维护一个加性误差为 ϵ 的解。预处理时间为 $O(1)$，而空间使用量仅为 $O(1/\epsilon)$，相对于向量 x 和 y的规模而言是次线性的。

▶ **定理 44**。设 $\epsilon > 0$。存在一种针对 ℓ_∞-项链对齐问题的静态算法，可在时间 $O(n+(1/\epsilon)^2\log(1/\epsilon))$内计算出加性误差为 ϵ 的解。此外，还存在一种针对 ℓ_∞-项链对齐问题的全动态算法，可在更新时间 $O(1/\epsilon^2\log(1/\epsilon))$ 内维护加性误差为 ϵ 的解，并且预处理时间为 $O(1)$；该算法的空间使用量为 $O(1/\epsilon)$。

为了得到动态算法的结果，我们证明：对于正在经历元素插入和删除的向量 $A, B \in \mathbb{R}^n$，可以动态维护（min $+$）-卷积 $A \oplus B$的一个近似。我们预计这一结果还会有更多应用。该定理的证明由下文的命题45和48得出。

G.2.1 静态算法

现在我们来考察静态算法，并证明以下命题。

▶**命题 45.** 存在一种用于 ℓ_∞-项链对齐问题的静态算法，能够在 $O(n+(1/\epsilon)^2\log(1/\epsilon))$时间内计算出加性误差为 ϵ 的解。

本小节余下部分将用于证明该命题。

算法。我们的算法相当简单，除了执行舍入的部分之外，与 Bremner 等人 [14] 使用的算法相同。考虑输入 ϵ（作为误差参数）、$x = \langle x_0, x_1, \ldots, x_{n-1}\rangle$ 和 $y = \langle y_0, y_1, \ldots, y_{n-1}\rangle$。现在我们设定 $\delta = \epsilon/2$，并对 x 和 y 各进行一次遍历，对每个条目应用舍入函数 $\lfloor\cdot\rfloor^*_\delta$。在此过程中，我们计算 x 和 y 的列表表示（其中将x和 y视为从 $[0,n$ 到 $[0,1)$) 的函数），它们至多包含 $O(1/\delta)$ 个分段（通过应用引理 39，取 $W=1$）。然后我们计算这些向量

$$x' = \langle x_0, x_1, \ldots, x_{n-1}, \underbrace{\infty, \ldots, \infty}_{n \text{ times}}\rangle,$$
$$x'' = \langle x_0, x_1, \ldots, x_{n-1}, \underbrace{-\infty, \ldots, -\infty}_{n \text{ times}}\rangle,$$
$$y' = \langle y_{n-1}, y_{n-2}, \ldots, y_0, y_{n-1}, y_{n-2}, \ldots, y_0\rangle,$$

但我们不显式存储它们，而只存储它们的列表表示。注意，x' 是一个单调递增向量，x'' 有两个单调递增段，而 y' 有两个单调递减段。

接下来，我们将 a 设为 x' 和 y' 的(min $-$)-卷积，并将 b 设为 x'' 和 y' 的(max $-$)-卷积（我们将在下文的引理 46 中证明，可以高效地计算这些函数）。

最后，我们设定 $v = \frac{1}{2}(b-a)$，并返回 $\min\{v_s\colon s \in [n]\}$ 作为问题的解。注意，我们可以通过 a 和 b 的列表表示高效地计算 v，并且还可以遍历 v的列表表示，快速求出 v_s 中的最小值。

分析。现在我们来分析上述算法。我们改编 Bremner 等人 [14] 关于近似解的定理 6 的证明，并说明如何使用分段常值函数来实现该算法。

我们首先说明，计算 (min $-$)-卷积和 (max $-$)-卷积的效率可以与经典的 (min $+$)-卷积一样高。

▶**引理 46**。设 f 和 g 是两个分段常值函数，分别有 p 个分段，并假设 g 有 k 个单调递减段。若当 f'和 g'有 k个单调递减段时，我们能在 $t(p,k)$ 时间内计算 f'和 g'的 $(\min +)$-卷积，那么我们就能在 $O(t(p,k) + p\log p)$ 时间内计算：

- $-(\max -)$-卷积：若 f 具有 k 个单调递增段，则 f 与 g 的卷积为此形式。若
- f 具有 k 个单调递增段，则 f 与 g 的 $(\min -)$- 卷积为此形式。

证明。首先，假设我们想要计算两个函数 $(\max -)$ 的卷积f 与 g。我们将说明，可以通过 (min $-$)-卷积 f 与 g 来计算 $(\min +)$-卷积 $-f$ 与 g。确实，对所有 x，都有：

$$\begin{aligned}\max_{\bar{x}\in[0,x]}\{f(\bar{x}) - g(x-\bar{x})\} &= \max_{\bar{x}\in[0,x]}\{-(-f(\bar{x}) + g(x-\bar{x}))\}\\ &= -\min_{\bar{x}\in[0,x]}\{-f(\bar{x}) + g(x-\bar{x})\}\\ &= -(((-f)\oplus g)(x)).\end{aligned}$$

为确认运行时间正确，注意到我们可以在 $O(p)$ 时间内计算 $-f$ 的列表表示，而更新存储 $-f$ 分段的二叉搜索树需要 $O(p\log p)$ 时间。此外，由于 f 具有 k 个单调递增段，因此 $-f$ 具有 k 个单调递减段。因此，我们可以在 $t(p,\ k)$ 时间内，对 $-f$ 和 g 应用高效的 (min $+$)-卷积算法。

我们可以通过计算一个 (min $+$)，类似地证明 (min $-$)-卷积的结果 - g 和 $-f$ 的卷积。更具体地说，对所有 x，都有

$$\min_{\bar{x}\in[0,x]}\{f(\bar{x}) - g(x-\bar{x})\} = \min_{\bar{x}\in[0,x]}\{-f(\bar{x}) + g(x-\bar{x})\} = ((-f)\oplus g)(x),$$

其中第一步用到了 (min $-$)-卷积的对称性。运行时间分析与上述完全相同。◀

在命题 45 的证明中，我们需要以下引理。我们将用该引理为给定的位移 s 找到最优偏移量 c。

▶**引理 47**（[14] 中的事实 5）。设 $z = \langle z_0,\ z_1,\ \ldots,\ z_{n-1}\rangle$。则

$$\min_{c\in\mathbb{R}}\max_{i=0}^{n-1}|z_i + c| = \frac{1}{2}\left(\max_{i=0}^{n-1} z_i - \min_{i=0}^{n-1} z_i\right)$$

而该量的极小值点为 $c = -\frac{1}{2}(\min_{i=0}^{n-1} z_i + \max_{i=0}^{n-1} z_i)$。

接下来，我们可以证明定理 44。

定理 44 的证明。我们分三步证明该定理。第一步，我们将证明在精确情形下（即不进行任何舍入时），我们能够计算出正确结果。这部分内容基本上沿用了 Bremner 等人的证明。 [14] 但会提供更多细节。第二步，我们讨论该算法的近似保证。第三步，我们证明其运行时间。

步骤 *1*：精确情形。首先，我们使用 Bremner 等人的定理 2 [14]，该定理指出，如果

$$\tilde{y} = \langle y_0, y_1, \ldots, y_{n-1}, y_0, y_1, \ldots, y_{n-1} \rangle$$

then

$$\min_{c,s} \max_{i=0}^{n-1} d^{\circ}((x_i + c) \mod 1, y_{(i+s) \mod n}) = \min_{c,s} \max_{i=0}^{n-1} d^{-}(x_i + c, \tilde{y}_{i+s}),$$

其中 $d^{-}(a, b) = |a - b|$ 对所有 $a, b \in \mathbb{R}$均成立。因此，我们不直接优化原始目标函数 $\min_{c,\, s} \max_{i=0}^{n-1} d^{\circ}((x_i + c) \bmod 1,\ y_{(i+s) \bmod n})$，而是考虑更方便的目标函数 $\min_{c,\, s} \max_{i=0}^{n-1} d^{-}(x_i + c,\ \tilde{y}_{i+s})$，它不涉及模运算。

事实上，考虑新的目标函数，并对所有 $s \in [n]$ 定义向量$z(s) \in \mathbb{R}^n$，使得 $z(s)_i = x_i - y_{(i+s) \bmod n}$。现在，对于新的目标函数，我们得到：

$$\begin{aligned}
\min_{c,s} \max_{i=0}^{n-1} d^{-}(x_i + c, \tilde{y}_{i+s}) &= \min_{c,s} \max_{i=0}^{n-1} \left| x_i + c - y_{(i+s) \mod n} \right| \\
&= \min_{s} \min_{c} \max_{i} |z(s)_i + c| \\
&= \min_{s} \frac{1}{2} \left(\max_{i} \{z(s)_i\} - \min_{i} \{z(s)_i\} \right),
\end{aligned}$$

其中，在第一步中，我们使用了 $d^{-}(\cdot,\cdot)$ 的定义，以及对所有 $k \in [2n]$ 均有 $\tilde{y}_k = y_{k \bmod n}$；在第二步中，我们代入了 $z(s)_i$ 的定义；在第三步中，我们应用了引理 47。

上述内容意味着，我们需要高效地计算 $\max_i\{z(s)_i\}$ 和 $\min_i\{z(s)_i\}$。进一步地，考虑向量 $v \in \mathbb{R}^n$，其各个分量为 $v_s = \frac{1}{2}(\max_i\{z(s)_i\} - \min_i\{z(s)_i\})$，并注意到上述计算表明，最优目标函数值等于 v 中最小的分量。因此，下面我们将证明，可以利用算法中计算得到的向量 a 和 b 高效地计算 v。

回顾两个向量 x' 和 y' 的定义：

$$x' = \langle x_0, x_1, \ldots, x_{n-1}, \underbrace{\infty, \ldots, \infty}_{n \text{ times}} \rangle,$$

$$y' = \langle y_{n-1}, y_{n-2}, \ldots, y_0, y_{n-1}, y_{n-2}, \ldots, y_0 \rangle.$$

现在，令 $a \in \mathbb{R}^{2n}$ 为 x' 和 y' 的 $(\min, -)$-卷积所得的向量，即 $a_k = \min_i\{x'_i - y_{k-i}\}$ 对所有 $k \in [2n]$ 均成立。现在我们注意到，对于每个满足 $s' \in [n]$ 的分量 $a_{n+s'}$，都有

$$a_{n+s'} = \min_{i=0}^{n+s'} \{x'_i - y'_{n+s'-i}\} = \min_{i=0}^{n-1} \{x_i - y_{(i-s'-1) \mod n}\},$$

其中在第二步中，我们利用了 $x'_i = \infty$ 对 $i \geq n$ 成立，以及 $y'_{n+s'-i} = y_{((n-1)-(n+s'-i)) \bmod n} = y_{(i-s'-1) \bmod n}$，因为在 y' 中，我们将 y 的各项按相反顺序拼接了两次。现在注意，如果 $s' = n - 1 - s$，那么

$$a_{2n-s-1} = a_{n+s'} = \min_{i=0}^{n-1} \{x_i - y_{(i-s'-1) \mod n}\} = \min_{i=0}^{n-1} \{x_i - y_{(i+s) \mod n}\} = \min_{i=0}^{n-1} \{z(s)_i\}.$$

接下来，我们定义向量 x''，使得：

$$x'' = \langle x_0, x_1, \ldots, x_{n-1}, \underbrace{-\infty, \ldots, -\infty}_{n \text{ times}} \rangle.$$

我们令 b 表示 x'' 和 y' 的（max $-$）卷积所得的向量，即对于所有 $k \in [2n]$，$b_k = \max_i\{x''_i - y'_{k-i}\}$。现在，采用与上文类似的论证可得，对于所有 $s \in [n]$，$b_{2n-s-1} = \max_{i=0}^{n-1}\{z(s)_i\}$。更具体地，对于满足 $s' \in [n]$ 的每个条目 $b_{n+s'}$，都有

$$b_{n+s'} = \max_{i=0}^{n+s'}\{x'_i - y'_{n+s'-i}\} = \max_{i=0}^{n-1}\{x_i - y_{(i-s'-1) \mod n}\},$$

这里我们用到了：对于 $i \geq n$，$x''_i = -\infty$，以及上文中关于 y' 和 y 各条目之间关系的相同论证。因此，若 $s' = n-1-s$，则

$$b_{2n-s-1} = b_{n+s'} = \max_{i=0}^{n-1}\{x_i - y_{(i-s'-1) \mod n}\} = \max_{i=0}^{n-1}\{x_i - y_{(i+s) \mod n}\} = \max_{i=0}^{n-1}\{z(s)_i\}.$$

结合以上结果可得，对于所有 $s \in [n]$，$v_s = \frac{1}{2}(b_{2n-s-1} - a_{2n-s-1})$。因此，最优目标函数值为 $\min_s v_s = \min_s \frac{1}{2}(b_{2n-s-1} - a_{2n-s-1})$。换言之，要计算最优目标函数值，只需计算差值 $\frac{1}{2}(b-a)$，然后返回 v 中索引介于 n 和 $2n-1$ 之间的最小条目。

步骤 *2*：近似保证。我们证明该算法返回一个加性ϵ-近似。首先注意，除了开头的舍入外，算法中的所有计算都是精确完成的。在舍入过程中，每个条目最多减少 $\delta = \epsilon/2$。因此，根据三角不等式，当我们将珠子 x_i 与珠子 y_{i+s}匹配时，近似引入的误差至多为 $2\delta = \epsilon$。由于目标函数只关心所有匹配珠子中的最大误差，这意味着我们得到一个加性 ϵ-近似。

步骤 *3*：运行时间分析。接下来需要分析该算法的运行时间。遍历输入向量 x 和 y、对条目进行舍入，以及计算 x 和 y 的列表表示，都可以在 $O(n)$时间内完成。回想一下，x 和 y 有 $O(1/\delta)$个分段。因此，我们也可以在 $O(1/\delta \log 1/\delta)$时间内计算向量 x'、x''和 y。接着，根据引理 46 和 40，我们可以分别在 $O(1/\delta^2 \log(1/\delta))$时间内计算 (min $-$)-卷积和 (min $+$)-卷积，且所得向量有 $O(1/\delta^2)$个分段。最后，向量 v 可以在 $O(1/\delta^2 \log(1/\delta))$时间内计算，而我们返回的最小值只需遍历 v的各个分段即可找到。由于我们此前已设定 $\delta = \epsilon/2$，证明至此完成。◀

G.2.2 动态算法

现在，我们将 ℓ_∞-项链对齐问题扩展到动态设置，其中可以对 x 和 y 执行插入和删除操作。

▶**命题 48.** 设 $\epsilon > 0$。存在一种用于 ℓ_∞-项链对齐问题的全动态算法，能够维护一个加性误差为 ϵ 的解，更新时间为 $O(1/\epsilon^2 \log(1/\epsilon))$，预处理时间为 $O(1)$；该算法的空间使用量为 $O(1/\epsilon)$。

证明。 在预处理过程中，我们将 x 和 y 初始化为空向量，并将它们存储为分段常数函数（参见第 2 节），而 不 将它们显式存储为向量。此外，我们设置 $\delta = \epsilon/2$。这些操作可在 $O(1)$ 时间内完成。

接下来，考虑操作 插入(i, α, β)，该操作要求将 α 插入 x 的第 i 个位置，并将 β 插入 y 的第 i 个位置。由于我们处于近似设置中，因此不插入 α 和 β的精确值，而是将 $\lfloor \alpha \rfloor_\delta^*$ 插入 x的第 i 个位置，并将 $\lfloor \beta \rfloor_\delta^*$ 插入 y的第 i 个位置。我们通过操作 x 和 y的列表表示来执行这些插入操作。我们仅描述如何对 x执行这些操作，因为对y 执行的操作基本相同。

将 x 的列表表示记为 (X_0, Y_0) ……，(X_p, Y_p)，其中 p 是 x 的分段数。现在，我们遍历 x 的所有分段，并检查是否存在一个值为 $Y_j = \lfloor \alpha \rfloor_\delta^*$ 的分段。如果不存在这样的分段，我们就在适当的位置将 $(i, \lfloor \alpha \rfloor_\delta^*)$ 插入列表表示中。然后，我们找到满足 $Y_j > \lfloor \alpha \rfloor_\delta^*$ 的最小整数 j，并将所有$k \geq j$的 X_k 增加 1。直观地说，我们将所有大于$\lfloor \alpha \rfloor_\delta^*$的分段向右移动一个单位，以便为刚插入的元素腾出空间。

按照上述方法更新 x 和 y 后，我们只需运行静态算法，不执行初始化 x 和 y 的步骤。注意，由于我们假设每次插入后 x 和 y 仍保持有序，并且只将舍入后的条目插入 x 和 y，根据引理 39， x 和 y 的分段数始终不会超过 $O(1/\delta)$。现在，由于我们在上文中设定了 $\delta = \epsilon/2$，命题 45 的证明表明，我们能在 $O(1/\epsilon^2 \log(1/\epsilon))$ 时间内得到一个加性误差为 ϵ 的解。此外，请注意，由于我们没有显式存储 x 和 y （只存储由其列表表示所表示的舍入版本），空间使用量为 $O(1/\epsilon)$。

最后，我们指出，操作 删除(i) 可以通过类似上述的方法实现：先操作 x 和 y 的列表表示，从 x 和 y 中删除第 i 个条目，然后运行静态算法。◀

我们指出，按照命题 48 的证明所述，存储两个正在进行元素插入和删除的动态向量 x 和 y，还可以通过引理 40 高效地维护它们的 (min $+$)- 卷积 $x \oplus y$ 的近似。

H 省略的证明

H.1 引理 6 的证明

分别将 g 和 h 的列表表示记作 (x_1^g, y_1^g)，……，$(x_{p_g}^g, y_{p_g}^g)$ 和 (x_1^h, y_1^h)，……，$(x_{p_h}^h, y_{p_h}^h)$。回忆一下，这两个列表表示都存储在双向链表中，而 g 和 h 的分段存储在二叉搜索树中，因此对于所有 $x \in [0, t]$，我们都可以分别在 $O(\log p_g)$ 和 $O(\log p_h)$ 时间内计算 $g(x)$ 和 $f(x)$。

我们通过展示如何构造它们的列表表示，说明如何构造函数 $f_{\min}$、f_{shift}、f_{add} 和 f_{round}。

首先，考虑 $f_{\min}$。我们构造 $f_{\min}$ 的列表表示 $(x_1^{\min}, y_1^{\min}) \ldots$。我们方法的直觉是，$f_{\min}$ 的每个分段都必须以 g 和 h 的分段的某个起点或终点作为起点和终点。因此，我们将在所有点 x_i^g 和 x_j^h 处计算函数 $\min\{g, h\}$，并据此设置 $f_{\min}$；然后，如果 $f_{\min}$ 包含多个具有相同 $y_i^{\min}$-值的分段，我们就删除这些重复分段。更具体地说，我们考虑集合$X = \{x_1^g, \ldots, x_{p_g}^g, x_1^h, \ldots, x_{p_h}^h\}$，并按从小到大的顺序排列。现在，对于所有 $i = 1, \ldots, p_g + p_h$，我们将 $x_i^{\min}$ 设为 X 中第 i 小的元素。注意，在区间 $[x_{i-1}^{\min}, x_i^{\min})$ 上，$f_{\min}$ 必须取值为 $\min\{g(x_{i-1}^{\min}) \; h(x_{i-1}^{\min})\}$。因此，我们将 $y_i^{\min} =$ 设为 $\min\{g(x_{i-1}^{\min}) \; h(x_{i-1}^{\min})\}$。由此得到 $f^{\min}$ 的初始列表表示。接着，我们对 $f_{\min}$ 的列表表示进行“剪枝”，也就是说，按照 i 的递增顺序遍历所有数对 $(x_i^{\min}, y_i^{\min})$；如果 $y_{i-1}^{\min} = y_i^{\min}$，就从 $f_{\min}$ 的列表表示中删除数对 $(x_{i-1}^{\min}, y_{i-1}^{\min})$。注意，在

这一过程结束时， $y_i^{\min}$ 的所有取值两两不相交（因为函数 g 和 h 是单调的）。

为了证明对所有 $x \in [0,t]$， $f_{\min}(x) = \min\{g(x)\ h(x)\}$，我们注意到，对于所有 $x \in X$（其中 X 与上文段落中的含义相同），根据构造，我们已正确设置了集合 $f_{\min}(x)$。此外，在 $[0,t] \setminus X$ 中的所有连续区间上， g 和 h 都是常数，因此 $f_{\min}$ 也是常数。因此，对于所有 $x \in [0,t] \setminus X$， $f_{\min}(x)$ 也都已正确设置。

接下来，我们注意到 $f_{\min}$ 至多有 p_g+p_h 个分段，因为 X 最初至多由 p_g+p_h 个元素组成，此后我们只从 $f_{\min}$ 中删除了分段。此外，对 X 中的元素排序需要时间 $O((p_g+p_h)\log(p_g+p_h))$，而计算 $\min\{g(x_i^{\min})\ h(x_i^{\min})\}$ 需要时间 $O(\log(p_g)+\log(p_h))$。之后，我们只用时间 $O(|X|) = O(p_g+p_h)$ 对 $f_{\min}$ 的列表表示进行了一次遍历。因此，创建 $f_{\min}$ 的列表表示需要时间 $O((p_g+p_h)\log(p_g+p_h))$。最后请注意，要将元素 $x_i^{\min}$ 存储在二叉搜索树中，还需要额外的时间 $O((p_g+p_h)\log(p_g+p_h))$。

现在我们注意到， f_{add} 可以用与计算 $f_{\min}$类似的方法计算：函数 f_{add} 仅在 X 中的点处改变其函数值（其中 X 与上文相同）。因此，我们令 x_i^{add}为 X 中第 i小的元素，并设置 $y_i^{\mathrm{add}} = g(x_{i-1}^{\mathrm{add}}) + h(x_{i-1}^{\mathrm{add}})$，然后执行与上文相同的剪枝步骤。证明的其余部分与上文相同。

接下来，考虑 f_{shift}。我们构造 f_{shift}的列表表示（$x_1^{\mathrm{shift}}, y_1^{\mathrm{shift}}) \ldots, (x_{p_g}^{\mathrm{shift}}, x_{p_g}^{\mathrm{shift}})$。对于所有 $i = 1, \ldots, p_g$，我们设置 $x_i^{\mathrm{shift}} = x_i^g + c$和 $y_i^{\mathrm{shift}} = y_i^g$。正确性显而易见；从构造过程也可看出，分段数只有 p_g ，且所有操作都能在 $O(p_g\log(p_g))$时间内完成（因为我们仍需为 f_{shift} 的分段构造二叉树）。

最后，考虑 f_{round}。与之前一样，我们构造 f_{round} 的列表表示：$(x_1^{\mathrm{round}},\ y_1^{\mathrm{round}})$.....，$(x_{p_g}^{\mathrm{round}},\ x_{p_g}^{\mathrm{round}})$。对于所有 $i = 1,$ ....., p_g，我们设定 $x_i^{\mathrm{round}} = x_i^g$ 和 $y_i^{\mathrm{round}} = \lceil y_i^g \rceil_{1+\delta}$。之后，我们执行与构造 $f_{\min}$ 时相同的剪枝步骤。由于 g 取值于 $W_\infty = \{0\} \cup [1, W] \cup \{+\infty\}$，且 g 单调， f_{round} 至多能取 $2 + \lceil \log_{1+\delta}(W) \rceil$ 个不同值。同样，运行时间界来自这样一个事实：我们必须为 f_{round} 的各个分段构造二叉搜索树。

H.2 引理 7 的证明

令 (x_1^s, y_1^s).....，$(x_{p_s}^s, y_{p_s}^s)$ 为 f_s 关于 $s = 1$、2 的列表表示，其中 $p_s \le p$ 是 f_s的分段数。对于所有 $(i,j) \in \{1, \ldots, p_1\} \times \{1, \ldots, p_2\}$，我们创建配对 (y_i^1, y_j^2)，并对它们排序，使得 $y_i^1 + y_j^2$ 单调递增。我们按此顺序遍历所有配对，并在每次迭代中，将某些 x 值对应的函数值 $f(x)$ 设为 $y := y_i^1 + y_j^2$，其中 (y_i^1, y_j^2) 是该次迭代所处理的配对。这里，我们从较大的 x 值开始（此时 f 取最小值），并不断减小 x （而函数值随之增大）；换言之，我们从右向左在定义域 $[0,t]$ 上构造 f。更具体地，令 $x_{\max}$ 表示尚未设定函数值的最高 x 值（若所有函数值均已设定，则令其为 $-\infty$ ）。令 $x' = x_{i-1}^1 + x_{j-1}^2$。我们将所有 $x \in [x', x_{\max})$ 对应的函数值设为 y ，然后设定 $x_{\max} = x'$。对于 f 的每个新分段，我们记录该分段是由 f_1 中所用分段的索引 i 与 f_2 中所用分段的索引 j 合并得到的。随后继续下一次迭代，直到所有函数值都已设定。

下面两个命题说明该过程是正确的。

1. 每个 x 都被赋予一个函数值，该值至多等于正确值 $f(x)$。 为证明这一点，令 $x \in [0,t]$ 并回忆

$$f(x) = \min_{\bar{x} \in [0,x]} f_1(\bar{x}) + f_2(x - \bar{x}) \ .$$

令 $\bar{x}^*$ 表示上述表达式中取得最小值的 $\bar{x}$ 的值，并令 i^* 和 j^* 分别表示 $\bar{x}^*$ 和 $x-\bar{x}^*$ 在 f_1 和 f_2的列表表示中所属分段的索引。这意味着 $\bar{x}^* \in [x^1_{i^*-1}, x^1_{i^*})$ 和 $x-\bar{x}^* \in [x^2_{j^*-1}, x^2_{j^*})$。因此， $x \geq x' = x^1_{i^*-1} + x^2_{j^*-1}$。所以，在针对数对 $(y^1_{i^*},\ y^2_{j^*})$ 的迭代中或此前，过程会为 x赋予一个函数值。由于该过程按单调递增的顺序赋予函数值，因此可以保证所赋予的函数值至多等于正确值。

2. 所赋予的函数值 y 至少为正确的值 $f(x)$。假设在某次迭代中，我们将函数值 $y = y^1_i + y^2_j$ 赋给 $x \in [x', \ldots, x_{\max})$，其中 $x' = x^1_{i-1} + x^2_{j-1}$。我们有

$$
\begin{aligned}
f(x) &= \min_{\bar{x}\in[0,x]} f_1(\bar{x}) + f_2(x-\bar{x}) && \text{(definition)}\\
&\leq f_1(x^1_{i-1}) + f_2(x - x^1_{i-1}) && \text{(consider } \bar{x} = x^1_{i-1}\text{)}\\
&\leq f_1(x^1_{i-1}) + f_2(x' - x^1_{i-1}) && (x' \leq x,\ f_2 \text{ monotonically decreasing})\\
&= f_1(x^1_{i-1}) + f_2(x^2_{j-1})\\
&= y^1_i + y^2_j && (y^1_i = f_1(x^1_{i-1}) \text{ and } y^2_j = f(x^2_{j-1}))\\
&= y\ .
\end{aligned}
$$

因此，所赋的值至少为 $f(x)$。

注意，我们可以在 $O(p^2 \log p)$ 时间内实现上述过程：首先，我们用 $O(p^2 \log p)$ 时间对至多 p^2 对进行排序。然后，每次迭代都能在常数时间内完成，因为将 $x \in [x', x_{\max})$ 的函数值设为 y，只需将数对 $(x_{\max},\ y)$ 添加到 f 的列表表示中，并将 $x_{\max}$ 更新为 x'，耗时 $O(1)$。

最后，假设我们已经计算出 f，并且给定 $x \in [0,t]$后，需要返回一个值$\bar{x}^* \in [0,t]$，使得 $f(x) = f_1(\bar{x}^*) + f_2(x-\bar{x}^*)$。首先，令 $(x_1, y_1) \ldots, (x_p, y_p)$ 表示 f 的列表表示。然后，由于我们将 f 中的数对 (x_i, y_i) 存储在二叉搜索树中，因此可以在 $O(\log p)$ 时间内确定 f 的分段 ℓ，使得 $x \in [x_\ell, x_{\ell+1})$。回想一下，对于 f 的每个分段，我们都存储了 f_1 和 f_2 中对应分段的索引 i 和 j，并将这些分段合并。现在注意，我们有 $\bar{x}^* \in [x^1_{i-1}, x^1_i)$ 和 $x-\bar{x}^* \in [x^2_{j-1}, x^2_j)$，其中 i 和 j 使得 f_1 和 f_2 中的这些分段构成 f 中对应的分段。因此，为了找到 $\bar{x}^*$，我们可以先尝试设定 $\bar{x}^* = x^1_{i-1}$。如果 $x - \bar{x}^* = x - x^1_{i-1} \in [x^2_{j-1}, x^2_j)$，则任务完成。否则，必有 $x - x^1_{i-1} \geq x^2_j$。因此，我们必须将 $\bar{x}^*$ 的值从 x^1_{i-1} 增大，直到其足够大，使得 $x - \bar{x}^* \in [x^2_{j-1}, x^2_j)$。这可以通过设定 $\Delta = (x - x^1_{x-1}) - x^2_j$，并令 $\bar{x}^* = x^1_{i-1} + \Delta + \frac{1}{2}$ 为 $\min\{x^1_i - (x^1_{i-1} + \Delta)\ x^2_j - x^2_{j-1}\}$ 来实现。注意， $\bar{x}^*$ 的这个值可以在 $O(1)$ 时间内计算。因此，返回 $\bar{x}^*$ 所需的总时间为 $O(\log p)$。

H.3 定理 9 的证明

回顾一下，依赖图是一个有向无环图。我们将没有任何入边的顶点称为叶节点。顶点 u 的层级 是从叶节点到 u 的最长路径的长度。注意，由于每个节点最多只能到达 h 个其他节点，因此每个顶点的层级至多为 h。

我们以自底向上的方式计算动态规划，从有向无环图的叶节点开始，然后递归地计算行 i 的解，其中 In(i) 的解已经计算完成。我们使用单调分段常数函数来存储近似解 ADP(i,·)。

我们通过对 i 在依赖图中的层级进行归纳法来证明该定理。我们证明一个更强的结论：对于层级为 ℓ 的每个 DP 行 i，ADP(i, ·) 都是 DP(i, ·) 的 $\alpha^{\ell+1}$-近似。

我们从叶节点（即层级为 0 的顶点）开始。对于叶节点 i，利用性质 4(b)˜ 和 4(c)，可得 $\mathcal{P}_i$ 返回 ADP(i, ·)；它是一个单调、分段常数函数，至多包含 p 个分段，并且是 DP(i, ·) 的 α-近似。

˜ ˜接下来，考虑一行 i，其层级为 ℓ。我们使用 P_i 来计算 ADP(i, ·) $=\mathrm{P}_i$({ADP(i', ·): $i' \in$In(i)}。根据归纳假设，所有解 ADP(i', ·)，$i' \in$ In(i) 都以单调分段常数函数的形式存储，且每个函数最多有 p 个分段。由于我们只应用引理 6 中的操作 O(1)次，分段数只会增长一个O(1)因子。由于我们至多只应用一次引理 7 中的 (min $+$)-卷积，卷积后的分段数至多为 $O(p^2)$。因此，我们处理的函数的分段数绝不会超过 $O(p^2)$。引理 6 和引理 7 给出的界意味着，计算 ˜P_i 所需的所有操作都能在至多 $O(p^2\log(p))$时间内完成。此外，根据归纳假设，且由于每个 i' 都处于层级 $\ell' \leq \ell - 1$，我们知道 ADP(i', ·) 是 DP(i', ·) 的一个 α^ℓ-近似。利用性质 (3) 和 4(a)，可得 ADP(i, ·) 是 DP(i, ·) 的一个$\alpha^{\ell+1}$-近似。

该定理的近似保证由性质 (2) 推出，它意味着$\ell \leq h$ 对于依赖图中的所有 DP 行 i 均成立。此外，我们在上文论证过，每个解 ADP(i, ·) 都可以在 $O(p^2\log(p))$ 时间内计算得出，因此总运行时间为$O(|I| \cdot p^2\log(p))$。

H.4 定理 10 的证明

考虑一个发生 DP(i, ·) 变化的行 i。注意，我们只需计算依赖图中从 i 可达的行 i' 对应的 DP 解。由于我们假设依赖图是有向无环图，且对于所有行 i，Reach(i) $\leq h$，因此这样的行最多有 h 个。在定理 9 的证明中，我们论证过，每个解 ADP(i, ·) 都可以在 $O(p^2\log(p))$ 时间内计算得出。这就证明了该定理。

H.5 Räcke 树的性质

设 $G=(V_G, E_G)$ 是一个无向图，$T=(V_T, E_T)$ 是 G 的一棵 Räcke 树。我们通过证明对于任意顶点集 $S_T \subseteq V_T$，均有 $\mathrm{cap}_T(S_T) \geq \mathrm{cap}_G(S)$ 来证明 $\mathrm{mincut}_T(A, B) \geq \mathrm{mincut}_G(A, B)$，其中 $S \subseteq V_G$ 是 V_T 中的叶节点集。

令 $S_T \subseteq V_T$，并考虑割 $(S_T, \bar{S}_T)$。我们用 S 表示将 S_T 限制在叶节点上的结果，并注意到 $(S, \bar{S})$ 也是 G 中的一个割。于是：

$$
\begin{aligned}
\mathrm{cap}_T(S_T) &= \sum_{(x_t,y_t)\in S_T\times\bar{S}_T} \mathrm{cap}_T(x_t,y_t) && \text{(definition of } \mathrm{cap}_T(S_T)) \\
&= \sum_{(x_t,y_t)\in S_T\times\bar{S}_T} \mathrm{cap}_G(V_{x_t}\cap V_{y_t}) && \text{(definition of tree edge capacity)} \\
&= \sum_{(x_t,y_t)\in S_T\times\bar{S}_T} \sum_{(x,y)\in V_{x_t}\times\bar{V}_{x_t}} \mathrm{cap}_G(x,y) && \text{(w.l.o.g. assume } V_{x_t}\subseteq V_{y_t}) \\
&= \sum_{\{x,y\}\in E_G} \mathrm{cap}_G(x,y) \sum_{(x_t,y_t)\in S_T\times\bar{S}_T} \mathbb{1}\{x\in V_{x_t} \wedge y\in \bar{V}_{x_t}\} && \text{（改变求和顺序）} \\
&\geq \sum_{(x,y)\in S\times\bar{S}} \mathrm{cap}_G(x,y) = \mathrm{cap}_G(S) \ .
\end{aligned}
$$

¯这里的不等式成立，是因为右边计入容量的一对 $(x, y) \in S \times S$ 对应于一条图边 $\{x, y\} \in E_G$ （位于 $x \in S$ 和 $y \in S$ 之间）。这

条图边对 T 中 x-y 路径上的每条边的容量都有贡献。这些边中必有一条被 S_T 割断，也就是说，对于这条树边，$\mathbb{1}\{x \in V_{x_t} \wedge y \in V_{x_t}\} = 1$。因此，其容量也计入了左边。

H.6 引理 18 的证明

下界是显然的，因为 $\mathcal{P}_i$ 是 $\mathcal{P}_i$ 的一个 α-近似。对于上界，我们对 ℓ 使用归纳法。对于 $\ell = 0$，注意到

$$\begin{aligned}
\mathrm{ADP}(i,\cdot) &= \mathcal{P}_i(\{\mathrm{ADP}(i') \colon\ i' \in \text{属于}(i)\}) && \text{(定义)}\\
&\le \alpha \mathcal{P}_i(\{\mathrm{ADP}(i') \colon\ i' \in \text{属于}(i)\}) && (\tilde{\mathcal{P}}_i \text{ 是 } \alpha\text{-近似})\\
&= \alpha \mathcal{P}_i(\emptyset) && (v_i \text{ 是叶节点})\\
&= \alpha \text{动态规划}(i) && \text{(该动态规划是良态的)}
\end{aligned}$$

对于 $\ell > 0$，我们有

$$\begin{aligned}
\mathrm{ADP}(i,\cdot) &= \mathcal{P}_i(\{\mathrm{ADP}(i') \colon\ i' \in \text{属于}(i)\}) && \text{(定义)}\\
&\le \alpha \mathcal{P}_i(\{\mathrm{ADP}(i') \colon\ i' \in \mathrm{In}(i)\}) && (\tilde{\mathcal{P}}_i \text{ 是 } \alpha\text{-近似})\\
&= \alpha \mathcal{P}_i(\{\alpha^{\ell} \text{动态规划}(i', \cdot) \colon\ i' \in \mathrm{In}(i)\}) && \text{(归纳假设)}\\
&= \alpha^{\ell+1} \text{动态规划}(i, \cdot) && \text{(该动态规划是良态的)}
\end{aligned}$$

这里的归纳假设利用了这样一个事实：所有 $i' \in \mathrm{In}(i)$ 的层级都严格低于依赖图中的 ℓ。

H.7 引理 19 的证明

关于近似保证的结论可直接由引理 18 以及根节点的层级至多为 h 这一事实得出（因为依赖树中最长的叶节点到根节点的路径长度为 h）。为获得运行时间 $O(|V_T| \cdot t)$，我们按此顺序计算解 $\mathrm{ADP}(v_1), \ldots, \mathrm{ADP}(v_n)$，即依据依赖有向无环图的拓扑序进行计算。根据行 i 的排序假设，并且由于所有 $\mathcal{P}_i$ 都可以在 t 时间内计算，结论得证。

H.8 引理 20 的证明

假设被插入或删除的边与顶点 i 相接。由于我们考虑的动态规划具有良好态，我们只需为依赖图中满足以下条件的顶点 j 重新计算动态规划解：存在一条从 i 到 j, $j \ge i$ 的有向路径。根据依赖图的构造，此类顶点至多有 h 个（因为依赖图中最长的叶节点到根节点的路径长度为 h）。因此，我们可以在 $O(h \cdot t)$ 时间内重新计算所有这些解。重新计算完成后，近似比保证由引理 19 得出。

H.9 引理 21 的证明

我们只证明所有函数 f_i 均为单调递减的情形。对于单调递增函数的情形，证明方法类似。令 $\mathcal{P}$ 表示函数 f_i 中所有分段的集合。考虑一个分段 $p \in \mathcal{P}$，它从 t_1 开始，到 t_2 结束，函数值为 α。我们构造一个分段常值函数 $f_p \colon [0,t] \to W_\infty$，它有两个分段，在 $[0,t_1)$ 上的函数值为 ∞，在 $[t_1,t]$ 上的函数值为 α（这意味着我们将该分段从 t_2 延伸至 t）。

由于这些函数是单调递减的，我们可以将 $f_{\min}$ 改写为分段函数 f_p的最小值，即：

$$f_{\min}(x) = \min_{p \in \mathcal{P}} f_p(x) \ .$$

现在，我们按起始点对 $\mathcal{P}$ 中的所有分段进行排序。按照排序顺序处理这些分段，即可逐步构建结果函数。令 f_{r-1} 表示对前 $r-1$ 个分段取最小值所得到的分段常值函数。为了计算 f_r，我们必须将 f_{r-1} 的最后一个分段与 $\mathcal{P}$中的第 r个分段 p_r 进行比较。如果 p_r 的值高于 $f_{r-1}(\infty)$（即 f_{r-1}中最后一个分段的值），则忽略分段 p_r。否则，我们将 f_{r-1} 当前最后一个分段的结束时间设为分段 p_r 的起始时间 t_r，并添加分段 p_r，包括其起始时间、值以及结束时间 t。运行时间主要由分段排序以及将它们添加到结果函数时插入二叉搜索树所需的时间决定。

H.10 引理 22 的证明

不失一般性，我们可以假设 f_2 是单调递减的（这可由（min $+$）-卷积的对称性推出）。现在，由以下计算可得该引理；其中在第三步中，我们利用了 f_2的单调性，即对所有 $x' \geq x$，都有 $f_2(x') \leq f_2(x)$：

$$\begin{aligned} f(x') &= \min_{\bar{x} \in [0, x']} f_1(\bar{x}) + f_2(x' - \bar{x}) \\ &\leq \min_{\bar{x} \in [0, x]} f_1(\bar{x}) + f_2(x' - \bar{x}) \\ &\leq \min_{\bar{x} \in [0, x]} f_1(\bar{x}) + f_2(x - \bar{x}) \\ &= f(x). \end{aligned}$$