最大化延迟的最优性数据结构（又称数据库破解）

抽象的

本文研究在事先不知道 $r \leqslant n$ 的值的情况下，如何以在线方式（即，仅在前一个查询的结果准备好后才给出下一个查询）最小化回答 n 个元素上的 r 个查询的总成本。传统的索引首先在 n 个元素上构建完整的索引，然后再回答查询，这可能并不合适，因为索引的构建时间 通常为 $\Omega(n \log n)$ 会成为性能瓶颈。相反，对于许多问题，对于每个 $r \in [1, n]$， r 个查询的总成本都有一个下限 $\Omega(n \log(1 + r))$。匹配这个下限是延迟数据结构(DDS) 的主要目标，在系统社区中也称为数据库破解。对于广泛的一类问题，我们提出了通用的归约，将传统索引转换为与r的长范围的下限匹配的 DDS 算法。对于可分解问题，如果可以在 $O(n \log n)$ 时间内构建数据结构并具有 $Q(n)$ 查询搜索时间，则我们的简化方法可产生一个在 $O(n \log(1 + r))$ 时间内运行的算法，其中对于所有 $n \log n$ $n \log n$，上限在温和约束下渐近最佳。$r \leqslant Q(n)$， $Q(n)$

特别地，如果 $Q(n) = O(\log n)$，则 $O(n \log(1 + r))$ 时间保证扩展到所有 $r \leqslant n$， 我们可以轻松地用它解决大量 DDS 问题。我们的结果可以推广到一类"频谱可索引问题"，它包含可分解问题类。

2012 ACM 主题分类计算理论→ 数据管理的数据结构和算法

关键词和短语延迟数据结构、数据库破解、数据结构

数字对象标识符10.4230/LIPIcs...

1 简介

传统索引首先在整个数据集上创建索引，然后才开始回答查询。例如，在 $O(n \log n)$ 时间内在由 n 个实值组成的（未排序）集合 S 上构建一棵二叉搜索树 (BST) 后，我们可以使用该树在 $O(\log n)$ 时间内回答每个前导查询[1]。然而，当数据集 S 仅被搜索少数几次时，此范例就不足为奇了。在极端情况下，如果只需要回答一个查询，那么最好的方法是完全扫描 S，这只需要 $O(n)$ 时间。更一般地，如果我们要以在线方式回答 r 个查询- 即在输出前一个查询的结果后给出下一个查询-则可能支付总成本 $O(n \log(1 + r))$ [19]。 在

当 $r \ll n$ （例如 $r = \text{polylog}\, n$ 或 $2^{\sqrt{\log n}}$）时，成本 $O(n \log(1 + r))$ 突破了 $\Omega(n \log n)$ 的障碍，这表明排序是不必要的。

上述情况发生在许多大数据应用中，这些应用收集了大量数据，但很少查询。这种现象促使系统社区开展了一项名为数据库破解的研究；参见 [12、13、16-18、23、29、31、32] 及其参考文献。在这些环境中，问题规模n非常大，以至于即使是排序也被认为是昂贵的，应该尽可能避免。此外，无法预测需要支持多少个查询（即前面提到的整数r） 。数据库破解不会立即构建完整的索引，而是在每次查询期间仅执行构建的计算部分。如果r最终

[1] 给定一个搜索值q，前趋查询将返回S中不超过q 的最大元素。

达到某个阈值时，索引将完全创建，但更有可能的是 r 会停止在某个明显低于该阈值的值。破解数据库的挑战在于确保“平滑度”：随着 r 的增长，到目前为止所有 r 查询的总成本应该尽可能缓慢地增加。

在理论领域，数据库破解的核心数据结构问题早在 1988 年就已由 Karp、Motwani 和 Raghavan [19] 以延迟数据结构(DDS)的名义进行了研究。对于一些问题，他们解释了如何在事先不知道 r 的情况下回答 $r \in [1, n]$ 个对 n 个元素的查询，总成本为 $O(n \log(1 + r))$。对于每个 $r \in [1, n]$，他们证明了这些问题的匹配下限为 $\Omega(n \log(1 + r))$。

通过简化，相同的下限已被证明适用于许多其他 DDS 问题。

本工作探讨了以下主题：如何设计一个通用的归约方法，给定个（传统)数据结构，可以自动将其转换为有效的 DDS 算法？

这种减少可能会显著简化 DDS 算法的设计，并为传统索引和 DDS 之间的复杂联系提供新的启示。

数学约定。我们使用 $\mathbb{N}^+$ 表示正整数集。对于任何整数 $x \geq 1$，让 $[x]$ 表示集合 $\{1, 2, \ldots, x\}$。默认情况下，每个对数的底数都是 2。定义 $\mathrm{Log}(x) = \log(1 + x)$，对于任何 $x \geq 1$。

1.1 问题定义

本小节将形式化要研究的问题。设 S 称为数据集，是从域 D 中抽取的一组 n 个元素。设 Q 是一个（可能无限的)集合，其中的元素称为谓词。给定谓词 $q \in Q$，对 S 发出的查询将返回一个答案，表示为 $\mathrm{Ans}_q(S)$。我们认为，对于任何 $q \in Q$，答案 $\mathrm{Ans}_q(S)$ 都可以用 $O(1)$ 个词表示，并且可以在 $O(n)$ 时间内计算出来（这本质上意味着可以使用穷举扫描等强力算法来回答查询）。

延迟数据结构化 (DDS)。我们现在将 DDS 问题形式化。最初，算法 A 提供数组中的数据集 S，其中元素任意排列。对手首先为第一个查询选择谓词 q_1，并从 A 中请求答案 $\mathrm{Ans}_{q_1}(S)$。迭代地，对于每个 $i \geq 2$，在获得第 $(i - 1)$ 个查询的答案之后，对手要么决定终止整个过程，要么为下一个查询选择谓词 q_i 并从 A 中请求 $\mathrm{Ans}_{q_i}(S)$。对手被允许观察 A 的执行情况，因此能够为 A 选择“坏” q_i。

如果对于每 $r \leq t$，前 r 个查询的处理总成本最多为 $\mathrm{Time}(n, r)$，则算法 A 可以保证 t 个查询的运行时间为 $\mathrm{Time}(n, r)$。我们将集中讨论 $t \leq n$，因为这是数据库破解的重要场景。

最优性。虽然上述设置对于 DDS 来说是“标准”，但就数据库破解而言，从另一个新角度进行调查是有意义的。

由于数据库破解主要在数据集接收相对较少的查询的情况下有用，因此必须设计在查询数量较少时特别有效的 DDS 算法。

为了将“特别有效”的概念形式化，我们利用这样一个事实：对于许多个 DDS 问题（包括本文中考虑的问题），存在硬度障碍，规定在相关计算模型下，对于每个 $r \in [1, n]$，$\mathrm{Time}(n, r) = \Omega(n\, \mathrm{Log}\, r)$。

受此启发，我们说如果算法 A 的运行时间对所有 $r \leq \mathrm{LogStreak}(n)$ 都满足 $\mathrm{Time}(n, r) = O(n\, \mathrm{Log}\, r)$，则算法 A 保证了 $\mathrm{LogStreak}(n)$ 的 $\log(r)$-streak。

最差的log(r)-streak 保证是LogStreak(n) = O(1) 这是微不足道的,因为任何查询都可以在O(n)时间内得到回答。理想情况下,我们希望LogStreak(n) = n,但这并不总是可能的,正如本文后面所讨论的。然而,对于实际使用来说,足以让算法确保LogStreak(n) = Ω(n) 其中某个常数 > 0 因为 Ω(n) 查询可能已经太多了,以至于数据库破解不再具有吸引力。

类问题。上述定义框架可以具体化为各种个问题实例,它们在数据域D、谓词域Q 和结构上有所不同 T。接下来,我们介绍两个与我们的讨论相关的问题类别:

- 如果对于任何不相交集S1, S2 ⊆ D和任何谓词,则问题实例是可分解的 q ∈ Q,可以从Ansq(S1)和Ansq(S2)中导出Ansq(S1 ∪ S2) 时间。

我们定义一个问题实例是(B(n), Q(n))谱可索引的,如果它具有对于每个数据集S ⊆ D,都满足以下属性:对于每个整数s ∈ [|S|],都有可能在O(|S| · B(s))时间内在S上构建一个数据结构,可以回答任何查询这 ($\frac{|S|}{s}$ · Q(s))时间。术语 "频谱可索引"被选择来反映构建就函数B(n)和Q(n)而言,"良好"的索引结构参数s 的整个频谱。

对于上述定义,有两点很重要:

- (B(n), Q(n))谱可索引性意味着我们可以在任何数据集S ⊆ D在O(|S| · B(|S|))时间内回答查询,时间为O(Q(|S|)) (对于这个目的,只需设置s = |S|)。
- 考虑具有以下属性的任何可分解问题实例:对于任何数据集 S ⊆ D,我们可以在O(|S| · B(|S|))时间内构建一个数据结构T ,以回答查询 O(Q(|S|))时间。那么,问题实例必须是(B(n), Q(n))谱可索引的。为了了解原因,给定一个整数s ∈ [|S|],将S任意分为m = |S|/s不相交子集S1, S2, ..., O(| Sm ,其中除Sm 外,所有子集的大小均为s 。对于每个i ∈ [m],创建 Si)中的结构T (Si) |· B(s))时间;创建所有m个结构的总时间是O(m · s · B(s)) = O(|S| · B(s))。要回答查询q,只需搜索每个T (Si)即可在O(Q(s)))时间内获得Ansq(Si) ,然后组合Ansq(S1)、 Ansq(S2)、 ...、 Ansq(Sm) 使用O(m)时间将查询结果转化为Ansq(S) 。因此,总查询时间为O(m · Q(s))。

1.2 相关工作

Motwani 和 Raghavan 在一篇会议论文 [24] 中提出了 DDS 的概念,被合并到与 Karp 合著的期刊文章 [19] 中。他们 [19] 设计针对以下 DDS 问题的种算法:

- 前置搜索,其中S由n 个实数组成,每个查询被赋予一个任意值q并返回S中q的前身。
- 半平面包含,其中S由R中的n 个半平面组成 2,并且每个查询都给出任意点q ∈ R 2 并返回q是否被S中的所有半平面覆盖。
- 凸包包含,其中S由R中的n 个点组成 2,每个查询都会被赋予任意点q ∈ R 2 并返回q是否被S的凸包覆盖。
- 二维线性规划,其中S由R中的n 个半平面组成 2,并且每个查询都给出二维向量u ,并返回所有n个半平面交点中的点p 最大化点积u · p。
- 正交范围计数,其中S由R中的n 个点组成 d 维数为d 是一个固定常数,查询是给定的任意d 矩形q 即

轴平行盒形式为[x1, y1] × [x2, y2] ×被 q 覆盖的S中的点。 ··· × [xd, yd] 返回

对于前四个问题，Karp、Motwani 和 Raghavan 提出了实现对于所有r ≤ n， Time(n, r) = O(n Log r) 。对于正交范围计数，他们提出了两个算法，其中第一个确保$\mathrm{Time}(n, r) = O(n \operatorname{Log}^d r)$ 对所有r ≤ n，而另一个确保$\mathrm{Time}(n, r) = O(n \log n + n \operatorname{Log}^{d-1} r)$，其中r ≤ n。对于所有这些问题，他们证明了，在比较模型和/或代数模型下，运行时间任何算法的都必须满足对于每个r∈ [1, n]，Time(n, r) = Ω(n Log r) 。

Aggarwal 和 Raghavan [1] 提出了一种 DDS 算法，其时间 (n, r) = O(n Log r) 对于所有r ≤ n 的最近邻搜索，其中S由R中的n 个点组成^2，并且查询是给定任意点q ∈ R^2 并返回S中距离 q最近的点。运行时间为对于所有r ≤ n ，在代数模型下都是最优的。

关于距离中值的 DDS 可以讲述一个“成功的故事” 。在问题的离线版本中，我们给定一个（未排序的)数组 A 中的n 个实数值集合S。对于任何 1 ≤ x ≤ y ≤ n，设A[x : y] 表示元素集{A[x], A[x + 1], ..., A[y]}。此外，给定r 个整数对(x1, y1),(x2, y2), ...,(xr, yr)，其中对于每个i ∈ [n]，1 ≤ xi ≤ yi ≤ n 。目标是找到集合A[xi : yi] 且其中所有i ∈ [r]。在 [15] 中，Har-Peled 和 Muthukrishnan 解释了如何在O(n Log r + r log n · Log r) 时间内解决这个问题，并证明了一个更低的在比较模型下，当r ≤ n 时， Ω(n Log r) 的界。在 [11] 中（另见 [5]） ，Gfeller 和 Sanders 考虑了该问题的 DDS 版本，其中S如前所述，并且查询给定一个任意对(x, y)，其中 1 ≤ x ≤ y ≤ n，并返回A[x : y]的中位数。他们设计了一种算法，当r ≤ n 时， Time(n, r) = O(n Log r) 。很容易看到任何 DDS 算法都可用于解决Time(n, r)中的离线问题次。因此，Gfeller 和 Sanders 的算法改进了 [15] 的离线解决方案，并且在比较模型下是最优的（对于 DDS）。

与我们的工作更相关的是 [8]，其中 Ching、Mehlhorn 和 Smid 考虑动态 DDS 问题，除了查询之外，还需要算法来支持次更新数据集S。在另一个方向上，Barbay 等人 [2] 研究了 DDS 版本的前身搜索，但使用更细粒度的参数分析了他们的算法称为“间隙”（而不是仅使用n和r）。这个方向也扩展到动态DDS；参见最近的著作[26, 27]。

如上所述，DDS 已在以下领域得到系统界的广泛研究：名称数据库破解。重点是设计有效的启发式方法来加速在各种实际场景中遇到的查询工作负载，而不是建立强大的理论保证。感兴趣的读者可以参考[12,13,16–18、23、31、32]作为进入文献的切入点。

1.3 我们的结果

我们的主要结果是具有以下保证的通用减少：

▶定理1.假设B(n)和Q(n)是非减函数，且B(n) = O（log n)且Q（n） = O（$n^{1-\varepsilon}$)其中 > 0是任意小的常数。每个(B(n), Q(n))频谱可索引问题允许使用 DDS 算法

$$\mathrm{LogStreak}(n) = 最小\{n, \frac{c \cdot n \log n}{Q(n)}\} \quad (1)$$

对于任意大的常数c，即算法实现Time(n, r) = O(n · Log r) 对于所有 $r \leq \min\{n, \frac{c \cdot n \log n}{Q(n)}\}$。

在温和的约束条件下,如我们在第3 节中论证的那样, (1)中的条纹界限是最好的。作为一个重要的特例,当$Q(n) = O(\log n)$时,由我们的归约产生的 DDS 算法实现$LogStreak(n) = n$,即,对于所有$r \leqslant n$, $Time(n, r) = O(n \cdot \log r)$ 。对于许多对数据库系统具有重要意义的可分解问题,构造时间为$O(n \log n)$,搜索时间为$O(\log n)$ 的数据结构已经已知(例如,前驱搜索和最近邻搜索)。由于这类问题必须是($\log n$, $\log n$)谱可索引的(如第 1.1 节所述),定理 1 立即为所有这些问题的 DDS 版本产生了优秀的解决方案,这些解决方案通常在比较/代数模型下是最优的。第 4.1 节将给出这些问题的一部分。

定理 1 为数据库破解带来了好消息:即使查询时间很慢的数据结构也可用于破解!例如,对于 2D 空间中的正交范围计数(在 1.2 节中定义),kd 树的构建时间为$O(n \log n)$,可在$Q(n) = O(\sqrt{n})$ 时间内回答查询。定理 1 表明,该结构可用于在$O(n \log r)$ 时间内回答任何$r = O(\sqrt{n} \log n)$ 查询,这对于现实中的数据库破解来说可能绰绰有余。这很好地体现了我们研究"对小r特别有效的 DDS 算法"的动机(在 1.1 节中说明)。

定理 1 对DDS 算法的设计具有指导意义 我们应该研究底层问题实际上的频谱可索引性。在这个方向上的探索会变得相当有趣,正如我们将在第 4.2 节中展示的半平面包含、凸包包含、范围中位数和二维线性规划(参见第 1.2 节了解它们的定义)。我们可以证明,对于适当选择的函数$Q(n)$,这些问题中的每一个都是($\log n$, $Q(n)$)频谱可索引的,然后利用定理 1 获得一个算法,该算法对所有$r \leqslant n$ 都使用 $Time(n, r) = O(n \log r)$ 来解决它。

构建需要$\omega(n \log n)$ 时间的结构,或者说$B(n) = \omega(\log n)$,会发生什么情况?第 5 节将展示,在温和的约束下,没有通用的归约可以使用这样的结构来获得$LogStreak(n) = \omega(1)$ 的 DDS 算法。换句话说,这些算法只能在$r = O(1)$的简单场景中实现$Time(n, r) = O(n \log r)$ 。

尽管如此,如果人们接受具有"对于所有$r \leqslant n$, $Time(n, r) \leqslant n \log^{O(1)}$ r"形式保证的算法,那么只要$B(n)$和$Q(n)$是$\log^{O(1)} n$,我们就可以扩展定理 1 所依据的归约来产生这样的算法,正如第 5 节要讨论的那样。

2 热身:前任搜索

前因搜索是 DDS 和数据库破解社区最关注的问题。本节将回顾 [19] 中开发的两种方法,用于实现此问题的$Time(n, r) = O(n \log r)$。这些方法构成了几乎所有现有DDS算法的基础。

自下而上。回想一下,数据集S由n 个实数组成。我们假设 wlog, n是 2 的幂。在任何时候,集合S都被任意划分为不相交的子集(称为运行),每个子集的大小相同, $s = 2^i$,其中$i \geqslant 0$。每个运行都经过排序并存储在数组中。最初, s = 1,即运行包含S 的单个元素。随着时间的推移,运行大小s单调增加。每当s需要从 2 开始,对于某个值$j > i$,就会进行大修以构建新的运行。由于可以通过在$O(2^j$ 个大小为 $2 \cdot (j - i))$ 的运行中合并 2^{j-i}来获得大小为 2^j的运行,因此大修可以在$O(n \cdot (j - i))$ 的时间内完成。因此,如果当前运行大小为s,则生成历史上所有次运行的总成本为$O(n \log s)$。

我

这是 2 个单位

每次运行时只需执行二进制搜索即可回答(前一个)查询。
然而,为了控制成本,该算法确保当前大小s至少为

在处理第 i 个（i ⩾ 1)个查询之前，先计算i Log i 。如果不满足条件，则调用大修以将s增加到最接近的 2 的幂至少i Log i。此后，查询· log(i Log i)) = O(n/i)时间。如果我们将所有r 个查询都加起来，总和变为O(n) = O(n Log r)。由于最终运行大小s为 需要花费O(n $\frac{}{s}$ 对数) = O(n $\frac{}{我登录我}$ O(r Log r)，我们可以得出结论，该算法在O(n Log r) 时间内处理r 个查询。注意这仅在r Log r ⩽ n （最大化运行大小为n)时成立。但是，当r达到 n/ log n时，算法可以在O(n log n) = O(n log r) 时间内对整个S进行排序，并在O(log n) = O(log r) 时间内回答每个后续查询。因此，对于所有r ⩽ n，该算法实现 Time(n, r) = O(n Log r) 。

自上而下。此方法模仿以下在 S上构建二叉搜索树(BST) T的策略：(i) 找到 S 的中位数，并在中位数处将S拆分为S1和S2 ；(ii) 将中位数存储为根的键，然后在S1 （或S2)上递归构建根的左（或右)子树。该算法不是直接进行完整构建，而是在查询处理期间 "按需"构建T。

一开始，只有根存在，并且它处于未扩展模式。通常，未扩展节点u还没有子节点，但是与应该存储在其子树中的元素的子集Su ⊆ S 相关联。查询的回答方式与普通 BST 中相同，即遍历T的根到叶路径π 。主要区别在于，当搜索到π上的未扩展节点u时，算法必须先扩展它。扩展u意味着为 u 创建两个子节点，在中位数（u 的关键字)处拆分Su ，在中位数处将 Su 分成两部分，并将每部分与一个子节点相关联。此后， u变为扩展状态，其子节点处于未扩展模式。扩展需要O(|Su|)时间 （查找集合的中位数需要线性时间[4]）。

经过r 次查询后，二叉搜索树T只被部分构建，因为在查询处理过程中，只有r条从根到叶的路径上的节点被扩展。前Log r层节点[2] 的总扩展成本为O(n Log r)。对于每条从根到叶的路径π，位于Log r层的节点u 的扩展成本为O(n/r)，这决定了π上u的 个后代节点的总扩展成本。因此，除了前Log r层的节点外， T中所有其他节点的总扩展成本为r · O(n $\frac{}{r}$) = O(n) 。因此，该算法

对所有r ⩽ n实现 Time(n, r) = O(n Log r) 。

3 从数据结构到DDS算法的简化

第 3.1 节将自下而上的方法（在上一节中进行了回顾)扩展为通用归约，这可以产生具有较大log(r) 条纹边界的 DDS 算法，前提是满足关键要求 线性可合并性（即将定义）。虽然这个归约将被我们在定理1 下的最终归约（在第 3.2 节中介绍)所取代，但其讨论 (i) 加深了读者对该方法的强大功能和局限性的理解，并且 (ii) 阐明了在没有线性可合并性的情况下新想法的必要性。最后，第 3.3 节将建立一个硬度结果，以表明定理 1 中的条纹边界不再能得到显着改善。

3.1 第一个简化：推广自下而上的方法

本节将重点讨论可分解问题。对于任何数据集S ⊆ D，我们假设存在一个数据结构T (S)，它可以在O(Q(n))时间内回答任何查询。此外，

[2] 节点的级别是从该节点到根的路径上的边的数量。

结构是线性可合并的，即，对于任何不相交的S_1、 $S_2 \subseteq D$， $S_1 \cup S_2$上的结构可以在$O(|S_1| + |S_2|)$时间内从$T(S_1)$和$T(S_2)$构造出来。请注意，这意味着 $T(S)$可以在$O(n \log n)$ 时间内构建。

▶引理 2.对于一个可分解问题，存在一个线性可合并结构，其中)其中 > 0为常数，我们可以设计一个 DDS 算法 查询时间$Q(n) = O(n^{1-\varepsilon})$保证 来 $\mathrm{LogStreak}(n) = \min\{n, \frac{c \cdot n \log n}{Q(n)}\}$对于任意大的常数 c。

证明。我们假设 wlog， n是 2 的幂。与自下而上的方法一样，在任何时候，我们将S任意划分为大小相同的运行$s = 2^i$，其中$i \geqslant 0$。对于每个运行，在其中的元素上构建一个结构。初始运行大小s为 1。每当s增长到 2^j （对于某个值$j > i$)时，就会进行一次大修以构建大小为 2^j的运行 从 2 开始通过 。

线性可合并性，我们可以通过在$O(2^j$ 个大小为 $2 \cdot (j - i))$ 的运行中合并2^{j-i}的结构来构建大小为 2^j的运行的结构。通过类似于第 2 节中的分析，如果当前运行大小为s，则生成所有运行的结构的总成本

历史上出现过的问题的答案是$O(n \log s)$。

带有谓词q的查询通过搜索每个运行的结构来回答，然后将所有运行的答案组合成$\mathrm{Ans}_q(S)$。查询成本为$O(\frac{n}{s} \cdot Q(s))$。我们要求在回答第 i 个（ $i \geqslant 1$)查询之前，运行大小s必须满足

$$Q(s)/s \leqslant 1/i。 \quad (2)$$

如果这个要求不满足，我们将进行彻底检查，将s增加到满足(2)的最小次方。这确保第 i 个查询的回答成本为$O(n/i)$。因此，处理r个查询的总成本为$O(n \log r)$。

剩下的就是限制大修的成本。由于（2），最终的运行规模为 是 2 的最小幂，满足$s \leqslant n$ （运行大小不能超过 n)和$s/Q(s) \geqslant r$ （因为 (2)）。

由于$Q(s) = O(s^{1-\varepsilon})$，其中$\alpha > 0$ 为常数，我们知道$Q(s) \leqslant \alpha s^{1-\varepsilon}$/α.s由于我们的目标是找到 s 的上限，我们要求s满足$s/\alpha \geqslant r$ （比$s/Q(s) \geqslant r$更严格），或等效地$s \geqslant (\alpha \cdot r)^{1/\varepsilon}$。因此，如果$(\alpha \cdot r)^{1/\varepsilon} \leqslant n$，则我们可以声称$s = O(r^{1/\varepsilon})$，因此$O(\log s) = O(\log r)$，在这种情况下，所有大修总共需要$O(n \log r)$ 时间。

如果$(\alpha \cdot r)^{1/\varepsilon} > n$，则上述策略不起作用。但是，在这种情况下， $r > n^{\varepsilon}/\alpha$，即r已经是 n 的多项式。这激发了以下蛮力策略。当r达到 n^{ε}/α时 我们称之为捕捉点 我们只需在 $O(n \log n) = O(n \log r)$ 时间内在整个 S 上创建一个结构，并使用它来回答每个后续查询}。捕捉点之后的所有查询的总成本为$O(n \log n) = O(n \log r)$。因此，我们获得了一种算法，该算法保证对于所有$r \leqslant \min\{n, \}$， $c \cdot n \log n$ $\mathrm{Time}(n, r) = O(n \log r)$ 。

在$O(Q(n))$时间内直到$r = \min\{n, \frac{c \cdot n \log n}{Q(n)}$

$\overline{}$ Q(n) ◀

上述简化主要依赖于结构T是线性可合并的事实。在这种情况下，运行次数将变为 $\Omega(n \cdot \log(2^i))$，否则，创建大小为 2 的所有 大修的总成本将为 $\Omega(n \cdot \log^2 r)$。接下来，我们将介绍另一种可以削减 Log r因子的简化方法。

3.2 第二个归约：非线性可合并性

我们现在将删除第 3.1 节中的线性可合并性要求并建立定理 1。

回想一下，底层问题是$(B(n), Q(n))$谱，可用$B(n) = O(\log n)$来索引

并且Q(n) = O($n^{1-ε}$) 其中某个常数 > 0。目标是设计一个算法对于所有r ≤ min{ n , $\frac{c \cdot n \log n}{Q(n)}$ },其中c > 0 可以是任何常数。

假设 wlog， n是 2 的幂。我们的算法以epoch 为单位执行。在第 i 个（ i⩾1)时期的开始,我们设置

$$s = 2^{2^i}$$

并在O(n · B(s))时间内在S上创建一个结构T （即(B(n), Q(n))频谱可索引性所承诺的结构）。该结构使我们能够在O(n · Q(s))时间内回答任何查询。 $\frac{}{s}$

第i 个时期结束于 s/Q(s)个查询3在该时期内得到回答之后。这些查询要求总成本

$$\frac{s}{问} \cdot 这 \frac{n}{s} \cdot 问题 = O(n)。$$

从上文可以清楚看出,第 i 个 epoch 的总计算时间为O(n · B(s)) = O(n · $2^{时}$).作为n · $2^{时}$ 当i增加 1 时,其总成本翻倍,回答r 个查询的总成本为),其中h 是所需的 epoch 数。确切地说， h 的值是最小的 O(n · $2^{时h}$

整数h ⩾ 1 满足两个条件：

- C1： $2^{子^{離}}$ ≤ n （s的值不得超过n）；
- C2： $\sum_{i=1}^{h} \frac{2^{我2}}{Q(22i)至}$ ⩾ r （h 个周期可以回答的查询数量必须少为r)。

由于我们的目标是找到 h 的上限,因此我们用更严格的条件替换条件C2 :对于某些 $子^{離}$ /Q(22^{时h}) ⩾ r。因为Q(n) = O($n^{1-ε}$),已知Q(n) ≤ α · $n^{1-ε}$ 常数α > 0。由此,我们进一步将C2修改为更为严格的不等式：

$$\frac{2^{2^{时小}}}{一个 \cdot (22小时)^{1-ε}} = \frac{(22^{时h})}{一个} \geq r \Leftrightarrow 2^{子^{離}} \geq (a \cdot r)\ 1/ \quad 。 \tag{3}$$

令H表示满足 (3) 式的最小整数h ⩾ 1。这意味着

$$2^{H-1\,2} < (α \cdot r)\ 1/ \quad \Leftrightarrow 2^{气^{氢}} < (α \cdot r)\ 2/ \quad 。 \tag{4}$$

当 2 $2^{赫}$ ≤ n,上述论证保证了迭代次数h 最多≤ n,满足条件C1)。在这种情况下,所有),根据（4)可知其复杂度为O(n Log r)。H （不等式 2 个时期的总成$2^{赫}$ ≤ n意味着 2 2^h 本为O(n · 2^{时h}) = O(n ·2^{赫}

如果 2 > n,上述论证就不成立。然而,当这种情况发生时,我们从 (4)知道(α · r) 2/ > $2^{2^{赫}}$ > n,从而导致r > n /2/α。一旦r达到 n /2 /α 即捕捉点我们就会在O($n^{2^{赫}}$ log n) = O(n log r) 时间内在整个S上创建一个结构T ,并使用它回答O(Q(n))时间内的每个后续查询,直到r = min{n, $\frac{c \cdot n \log n}{Q(n)}$ }。捕捉点之后的 个查询需要的总成本为O(n log n) = O(n log r)。因此我们有

获得了一个算法,保证对于所有r ≤ min {n , $\frac{c \cdot n \log n}{Q(n)}$ 完成定理1的证明。

3.3 条纹边界的紧密性

本节将解释为什么对于具有“合理”行为的约简算法来说,条纹界限 Ω($\frac{对数n}{Q(n)}$)在定理 1 中渐近地是最好的。

---

[3] 在实践中,我们的算法可以通过使这个数字为 s · B(s)/Q(s)而略有改进,但这对于证明定理 1 来说并不是必要的

具有受限结构的可分解问题的黑盒约化。

为了证明难度结果，我们可以随意专门化问题类，我们通过仅考虑可分解问题来实现这一点。需要一种简化算法A 来处理任何 (B(n), Q(n))谱可索引的可分解问题。如第 1.1 节所述， 这意味着可以在O(|S| · B (|S|))时间内在任何S ⊆ D上构建一个数据结构T ，并在O(Q(|S|))时间内回答S上的任何查询。

只要保留(B(n), Q(n))谱可索引性，我们就可以限制功能

使得 A 难以解决。具体来说，T 只为 A 提供以下"服务"：给定谓词q ∈ Q，A 可以创建结构 T (S ;给定谓词q ，A 可以使用

T (S ;在A已经获得Ansq(S 1) 和Ansq(S)对任意子集S ⊆S ;

⊆ S 之后，它可以将答案组合成Ansq(S)找到答案Ansq（S)在S ;

2) 为不相交) 在恒

子集S 1、S_2' 1 ∪小号 2 定时间内 (

组合算法由 T 提供）。

尽管功能有限， T仍然使得问题(B(n), Q(n))谱可索引 ，因为该问题是可分解的；参见第 1.1 节中的解释。

到目前为止，还没有对 A 的行为施加任何限制，但现在我们准备这样做。要回答查询，算法A需要搜索数字 上的结构）。该算法可以选择任何（它们不需要不相交）。此外， A还被允许|)时间。有了所有这些，算法

（包括零)S 的子集，比如T (S t ⩾ 0 和任何S 1), T (S2),..., T (S 吨

另一个子集S S 1 2 , ..., t检查

扫描 ⊆ S ，支付Ω(|S 扫描

必须确保

$$年代扫描 ∪小号_1 ∪小号_2 ∪ ... ∪S_吨 = S. \quad (5)$$

上述约束是自然的，因为否则， S中至少有一个元素不存在于 中。如果算法 A "敢" 无论如何返回查询答案，那么它一定已经获得了底层问题的某些特殊属性。在这项工作中，我们关注的是

年代扫描 ∪小号 1 ∪小号 2 ∪ ... ∪S 吨 不知道问题特定属性的通用归约。

我们将满足上述要求的约简算法称为黑盒约简。引理 2 和定理 1 中的算法属于黑盒类。

定理1的严密性。我们将证明，对于任何函数Q(n) : N + → N + ，任何黑箱归约都只能在O(n Log r) 时间内回答n log n r = O() 个查询，并且Q(n)满足Q(n) = O(n)，且对于任何常数 c > 0，

Q(n) = Θ(Q(c · n)) ；是亚可加性的，即，对于任何x,y ⩾ 1 ， Q(x + y) ⩽ Q(x) + Q(y)成立。

这将证实定理 1 对黑盒类的紧密性。

我们将设计一个可分解的问题和一个附带的数据结构，这两者在现实中都没有意义，但在数学上却是合理的。数据集S 由n 个任意元素组成；给定任何谓词，对S的查询始终返回|S| （元素和谓词的具体形式无关紧要）。每当被要求在 S 上 "构建"数据结构时，我们都会故意浪费|S| log |S|时间，然后简单地输出数组中S的任意排列。每当被要求 "回答"查询时，我们都会故意浪费 Q(|S|)时间，然后返回|S|。该问题显然是可分解的。

我们认为，任何黑盒归约算法A都需要Ω(Q(n)) 的时间来回答每个查询。考虑一个任意查询，假设A通过搜索结构T (S 1), ..., T (S 来处理它，查询成本至少为

吨) 其中t ⩾ 0 且扫描S 扫描 .通过我们的结构设计，

$$哦(|S'_{扫描}|) + \sum_{我\in[t]} 问题(|S'_我|) \geqslant Ω(|S'_{扫描}|) + Q\left(\sum_{我\in[t]} |S'_我|\right) \quad （通过子可加性）$$

$$= \Omega\ Q(|S'_{扫描}|) + Q \underset{我\in[t]}{} |S'_{我}| \quad (由Q(n) = O(n), Q(n) = \theta(Q(cn)))$$

$$= \Omega\ Q\ |S'_{扫描}| + \underset{我\in[t]}{} |S'_{我}| \quad （通过子可加性）$$

$$= \Omega(Q(n)).\ （根据（5））$$

根据log(r)-streak的定义,算法A必须处理r = LogStreak(n) 个查询总成本为O(n log r),显然不能超过O(n log n)（记住r ⩽ n）。因此,A 只能处理O($\frac{对数n}{Q(n)}$) 查询。

4 个针对具体问题的新型 DDS 算法

我们现在部署定理 1 来开发针对具体 DDS 问题的算法,重点关注第 4.1 节中为个可分解问题,第 4.2 节中为不可分解问题。

4.1 可分解问题的应用

如上所述,如果可分解问题具有可以内置的数据结构T O(n log n) 时间（即B(n) = O(log n))并支持Q(n) = O(log n) 时间内的查询，定理 1 直接得出一个 DDS 算法,其中Time(n, r) = O(n Log r)（对于所有r ⩽ n）。我们下面列举了对数据库系统具有重要意义的问题的部分列表，之前没有发现有同样保证的算法。

- 2D 正交范围计数。问题定义见 1.2 节。结构 T 可以是持久的 “聚合”二叉搜索树（BST)[30]。
- 矩形上的正交范围计数。数据集S是一组n 个2 维矩形（即 R中的轴平行框 ^{2} 给定任意一个 2 矩形q,查询返回有多少个 S与 q 相交的矩形。该问题可以简化为四个查询 前一个问题（点的正交范围计数） [33]。T可以再次成为 持久性聚合 BST [30]。
- 点位置。数据集S是R的平面细分 ^{2} 由n 个线段定义，其中每个部分都与它所属的两个面的 ID 相关联。给定 任意点q ∈ R ^{2}，查询返回包含 q 的细分面，归根结底就是找到q正上方的线段（即第一个 “命中”的线段 射线从q 向上射出）。结构T可以是持久 BST [28] 或 柯克帕特里克结构[20]。
- 二维中k = O(1)最近邻搜索。数据集S是R中的一组n 个点 ^{2}. 修复 常数整数k ⩾ 1。给定一个点q ∈ R ^{2}，查询返回P中最接近的k 个点 q. 结构T可以是基于 k 阶Voronoi构建的点位置结构 [20, 28] 图（k 阶Voronoi 图可以在O(n log n) 时间内计算出来[6]）。当k = 1 时，已经发现了一个 DDS 算法,其中Time(n, r) = O(n Log r)（对于所有r ⩽ n） [1]。然而，[1] 中的算法很大程度上依赖于合并两个（1 阶)Voronoi 图的能力 在线性时间内,因此不能轻易扩展到更高的k值。
- 度量空间中的近似最近邻搜索。数据集S由n 个对象组成 在具有恒定倍增维度的度量空间中（这封装了任何欧几里得 维数为常数的空间。设dist(o1, o2)表示 空间中的两个对象o1和o2 。给定空间中的任意对象q ,查询

返回一个对象o ∈ S，使得dist(o, q) ≤ (1 +)·dist(o′, q) 对所有o′ ∈ S,其中是一个常数。在[14, 22]中可以找到满足我们目的的结构T。

对于R^{d}中的正交范围计数 其中d ≥ 3 是一个固定常数（定义在 1.2 节），我们可以应用定理 1 得到一个不太寻常的结果。有可能 [3] 在O(n log n) 时间内构建一个结构T，以回答Q(n) = O(n)中的查询) 时间 常数 > 0 可以任意小。定理 1 由此得到 DDS 算法 对于所有r ≤ n， Time(n, r) = O(n Log r)$^{1-ε}$。由于可以任意接近于 0，log(r)-条纹界限LogStreak(n) = n$^{1-ε}$ 比最大值n仅低 n中的因式子多项式。如果这个差距可以 对于所有常数维数都是封闭的。如果LogStreak(n) = n的 DDS 算法可以 被发现，那么该算法也将在O(n log n)中解决以下离线版本 时间：我们给定R中的n 个点的集合P^d 以及n 个d 矩形的集合Q；目标是 报告，对于每个d 矩形q ∈ Q， P中有多少个点在 q中被覆盖。此离线 问题已经得到广泛研究，但最快的算法仍然在n logΘ(d)中运行 n 据我们所知是次。

4.2 不可分解但可谱索引的问题

本小节将利用定理 1 来处理不可分解的问题 个问题（至少不是很明显）。关键是要表明问题是(Log n, Q(n)) 频谱可索引为合适的Q(n)。这本身就是一个有趣的话题，因为我们 接下来通过开发新的 DDS 算法来演示，其中Time(n, r) = O(n Log r) r ≤ n在半平面包含、凸包包含、范围中值和二维线性 编程。这些问题的原始算法 [11, 19] 都是使用 自上而下的方法在第 2 节中进行了回顾。我们的算法提出了一个对比，说明 定理1如何促进DDS算法的设计。

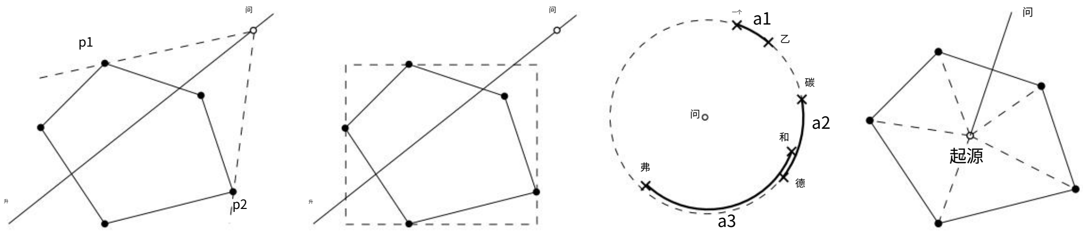
半平面约束。该问题可以转换为 [19] 等价于以下问题 形式，利用几何对偶[7]：

- 穿过凸包的线。S是R中的n 个点的集合^2. 给定R中的任意一行l^2，查询 确定l是否与 S 的凸包相交，表示为CH(S)。

我们将集中讨论上述问题。

现在假设我们给定l上的一个点q，该点位于CH(S) 之外。从q 开始，我们 可以发射两条切射线，每条都接触CH(S)，但不会进入 CH(S) 的内部。 在图 1(a) 中， S是黑点集，第一条射线经过点p1 ∈ S，而 第二次通过p2 ∈ S。这两条射线形成一个“楔形”，将CH(S)包围在其中（注意 楔角小于180° ）；我们称它为q在CH(S)上的楔。线l通过 通过CH(S)当且仅当l通过楔形。如果CH(S)的顶点已经 按顺时针顺序存储，可以在O（log n)时间内找到两个切线[25]。

我们将证明“通过凸包的线”问题是（Log n， Log n)谱 可索引。取任意整数s ∈ [1, n] 并设置m = n/s 。将S任意分为S1、 S2、 ...， Sm使得|Si | = s，其中i ∈ [m − 1] 且|Sm| = n − s(m − 1)。构建结构T ， 使用O(s Log s) 时间计算每个i ∈ [m]的CH(Si)，并按顺时针方向存储其顶点 顺序。该结构的构建时间为O(n Log s)。让我们看看如何回答查询 带有直线l。再次假设给定l上的CH(S)外的一个点q 。对于每个i ∈ [m]， 在O(Log s) 时间内计算CH(Si)上q的楔形。从这m 个楔形中，可以得到一个简单的 任务是在O(m)时间内获得CH(S)上的q的楔形（我们将处理一个更一般的 问题在讨论“凸包包含”时很快会得到解决）。现在， l是否与 CH(S)可以轻松确定。到目前为止，查询时间为O(m Log s)。



(a)切线射线 (二)MBR (c)弧线 (d) 解剖

图 1 4.2 节中关键概念的说明

剩下要解释如何找到q。如果我们已经具有 S 的最小轴平行边界矩形，表示为MBR(S)。注意 MBR(S)必定含有CH(S)；见图 1(b)。显然， MBR(S)可以得到在O(m)时间内从MBR(S1), MBR(S2), ..., MBR(Sm)生成，而每个MBR(Si) (i ∈ [m]) 都可以在结构T的构造过程中可以在O(s)时间内计算出来。因此我们得出结论问题是 (Log n, Log n) 谱可索引的，现在可以应用定理 1。

凸包包含。在这个问题中， S是R中的一组n 个点 2 . 给定任意点q ∈ R 2，查询确定q是否被CH(S) 覆盖。我们将证明问题是 (Log n, Log n) 频谱可索引的。

取任意整数s ∈ [1, n]，设m = n/s 。将S任意分为S1, S2, ..., Sm 使得|Si 其中i ∈ [m − 1] 且|Sm| = n − s(m − 1)，则 | = s 。要构建结构T ，计算 CH(Si)，对每个i ∈ [m]进行排序，并按顺时针顺序存储其顶点；这需要O(n Log s) 时间，因为之前解释过。让我们看看如何回答给定点 q 的查询。对于每个i ∈ [m]，是否 q是否被CH(Si)覆盖可以在O(Log s) 时间内得到检验[25]。如果对任意 i 的答案为是，点q一定被CH(S)覆盖，这样就完成了。接下来的讨论假设对于所有i， q 都在CH (Si)之外。在O(m Log s) 时间内，计算q在CH(Si)上的楔形，对于所有i ∈ [m]，如上一个问题所述。

剩下的就是根据m 个楔形确定q是否被 CH(S) 覆盖。这可以是重新建模为以下问题。以q 为中心放置一个任意圆。对于每个楔形，它的两条边界射线与圆相交成小于 180° 的圆弧 . 设a1, a2, ..., am为以这种方式产生的弧，并定义一个 是覆盖它们的圆上的最小圆弧全部。图 1(c) 显示了m = 3的示例。圆弧a1由点A和B所对，圆弧点C和D 组成圆弧 a2 ，点E和F 组成圆弧a3 。其中，最小圆弧a 顺时针从点A到F。至关重要的是，当且仅当 跨度至少 180 ◦ （这是图 1(c) 中的情况） 注意，如果q在CH(S) 之外，那么a必须被 q在CH(S)上的楔形，必须小于 180 ◦ 。因此，目标是确定是否 至少为 180 ◦ 。

可以在O(m)时间内实现目的（即使m 个弧以任意顺序）。为此，我们逐个处理圆弧，保持最小圆弧 → 覆盖已经处理的弧，并在清楚时停止算法 → 必须至少为 180 ◦ 。具体来说，对于i = 1，只需设置一个 到a1。迭代地，给定下一个圆弧ai (i ⩾ 2)，在常数时间内检查圆弧是否小于 180 ◦ 可以覆盖人工智能和 → 。如果是，请更新 到该弧；否则，停止算法。例如，在图 1(c) 中，当处理完a2之后，我们维护的a就从A顺时针转到D了。当处理a3的时候，该算法实现了 必须至少 180 ◦ 并因此终止。

因此我们得出结论，凸包包含问题是（Log n， Log n)谱可转位，现在可以应用定理 1。

范围中位数。在这个问题中， S是存储在数组A中的一组n 个实数。给定

满足 $1 \leqslant x \leqslant y \leqslant n$ 的任意整数对(x, y)，查询返回 $A[x : y]$。我们将证明该问题是$(\text{Log } n, \text{Log } n)$ 谱可索引的。修复任何整数$s \leqslant n$且$m = n/s$ 。定义$S_i = A[(i - 1)s + 1 : i \cdot s]$，其中$i \leqslant m - 1$ 且$S_m = A[(m - 1)s + 1 : n]$. 接下来，我们假设$s \leqslant \sqrt{n}$;否则，只需使用 $O(n \log n) = O(n \log s)$ 的时间来创建一个 [11] 的结构在整个S 上，它能够在$O(\log n) = O(\log s)$ 时间内回答S上的任何查询。

建立一个结构T ，对于每个$i \in [m]$，存储S_i 按升序排列，但每个元素 S_i的应该与它在 A 中的原始位置索引相关联。显然， T可以构建时间为$O(n \text{ Log } s)$。让我们看看如何回答带有谓词(x, y) 的查询。首先，确定值$a, b \in [m]$使得$A[x] \in S_a$且$A[y] \in S_b$，这可以简单地在$O(m)$时间内完成。扫描S_a并识别子集$S'_a = S_a \cap A[x : y]$（对于每个元素在 S_a中，检查它在A中的原始索引是否在$[x, y]$ 中。因为S_a是排序的，我们可以得出 S'_a 按排序顺序在$O(s)$时间内完成。以同样的方式，计算$S'_b = S_b \cap A[x : y]$ 在 $O(s)$时间。此时， $A[x : y]$ 的所有元素都已在$b - a + 1$中分割排序数组： $S'_a, S_{a+1}, S_{a+2}, \ldots, S_{b-1}, S'_b$。现在的目标是找到第 $(y - x + 1)/2$ 这些数组并集中的最小元素。Frederickson 和 Johnson [10] 描述了一种从排序数组的并集中选择给定等级的元素的算法。它们的在我们的场景中，算法运行时间为$O(m \text{ Log }) = O(m \text{ Log } s)$。总体查询时间为 $O(s + m \text{ Log } s) = O(m \text{ Log } s)$，因为$s \leqslant \sqrt{n}$.

我们现在得出结论，范围中位数问题是（Log n， Log n)谱可索引的并准备应用定理 1。

二维线性规划。利用几何对偶 [7]，该问题可以表示为将 [19] 转换为“穿过凸包的线”（我们已经解决了），如下所示：

- 射线离开凸包。其中， S是R^2中的一组n 个点使得CH(S)覆盖原点。给定从原点发出的任何射线q ，查询将返回边e_{exit} CH(S)，其中q退出CH(S)。

我们将集中讨论上述问题。在继续之前，让我们陈述两个事实关于问题：

- 给定任意射线q，都有可能找到e_{exit}，将此称为基本算法。 使用[21]中的算法，在O(n)时间内，我们
- 我们可以在$O(n \log n)$ 时间内创建一个结构，以在$O(\log n)$ 时间内回答任何查询。首先计算CH(S) ，然后使用连接原点和所有顶点;参见图 1(d)。射线q的查询可以通过查找 q经过的剖分三角形。我们将其称为基本结构。

与迄今为止讨论的所有问题不同，目前我们无法证明“射线退出凸包”是$(\text{Log } n, \text{Log } n)$ 谱可索引的。然而，根据定理 1，我们不必！只需表明问题是（Log n， n^{1-c}) 频谱可索引任何正常数$c < 1$。定理 1 允许我们回答$r \leqslant n^{1-c}$ 登录n查询 $O(n \text{ Log } r)$ 时间。当r达到n^{1-c}，我们有能力建立基本结构花费$O(n \log n) = O(n \log r)$ 的时间回答每个后续查询，时间为$O(\log n) = O(\log r)$。这使我们能够实现对于所有$r \leqslant n$ 的 $\text{Time}(n, r) = O(n \text{ Log } r)$ 。

我们将证明“射线离开凸包”问题是（Log n， $\sqrt{n}$) 频谱可索引。我们将通过 [19] 的结果来实现这一目的，其中 Karp、Motwani 和 Raghavan 采用自上而下的方法构建了具有以下属性的二叉树T。

- 如果节点u位于T 的层级ℓ，那么u与$n/2^{\ell}$的集合S(u)相关联 S中的点。
- 树的前ℓ层可以在$O(n \cdot \ell)$ 时间内构建。

一个查询的答案最多是遍历T的一条从根到叶的路径。如果搜索过程延伸到节点u，那么通过在 S(u)上运行基本算法 [21]，可以在O(|S(u)|)时间内找到目标边eexit 。

回到我们的场景，固定任何整数s ≤ n。在O(n Log √ s) = O(n Log s) 时间内在S上构建T的前 1 + Log √ s层。为了回答查询，如果尚未找到边eexit，我们将T的路径下降到层数为 Log √ s的节点u 。集合|S(u)|最多有n/2 Log √ s = n/√ s个点。因此，我们可以在S(u)上运行基本算法，在O(n/√ s) = O(n · √ s) 时间内找到eexit 。因此，“射线离开凸包”问题是(Log n, √ n) 谱可索引的。

s

我们用上面的讨论所揭示的关于自上而下的方法和我们的约简之间的内在联系的评论来结束本节。本质上，我们逐步构建[19] 的结构： T的第 i （i ≥ 1)个 “时期”（在第 3.2 节的证明中)级别。这不同于 [19] 中节点 “在第一次接触时展开”的方式，如第 2 节所述。事实上，所有现有的基于自上而下方法设计的 DDS 算法都可以通过 “谱可索引性”的桥梁封装到我们的约简框架中，就像我们为 “射线离开凸包”所展示的方式一样。

重建前 2 2^i

5 使用具有ω（n log n)构造时间的结构的DDS

到目前为止，我们的讨论集中在B(n) = O(log n) 的数据结构上。在本节中， 我们将首先说明黑盒归约的这一条件的必要性，以保证非常量log (r) 条纹边界。作为第二步，我们提出了定理 1 的扩展，该定理允许部署max{B(n), Q(n)} = polylog n的结构，以生成一个DDS算法，对于所有r ≤ n，该算法的Time(n, r) 都很好。

B(n) = ω(log n)的常数条纹边界。我们随后的硬度论证要求n · B(n)为凸函数。考虑任何黑盒归约算法A。

回想一下， A需要处理具有受限数据结构的任何可分解问题（读者可能希望在继续阅读之前查看第 3.3 节）。假设A可以保证所有此类问题的LogStreak(n)的 log(r)-streak 界限，即，对于所有r ≤ LogStreak(n)， A总是可以在O(n Log r) 时间内回答r 个查询。我们将证明，

如果B(n) = ω(log n)，则 LogStreak(n)必须是O(1)。

以类似于第 3.3 节的方式，我们将设计一个可分解的问题和一个伴随的数据结构。数据集S包含n 个任意元素；给定任何谓词，对S的查询始终返回|S|。每当被要求在S上构建数据结构T (S)时，我们都会故意浪费|S| · B(|S|)时间，然后在数组中输出S的任意排列。

每当被要求回答一个问题时，我们都会立即在常数时间内返回|S| 。换句话说，函数Q(n)固定为 1。

此后，我们将r固定为A在给定我们设计的数据结构时可以确保的LogStreak (n)的值。我们假设r = ω(1)；否则， LogStreak(n) = O(1) 并且我们的工作已完成。由于它在O(n Log r) 时间内回答了r 个查询，因此至少有一个查询）。我们将在其余部分集中讨论这个特定查询

必须有一个O(的参数成 n对数

本。

回想一下 3.3 节，为了回答这个查询，算法A需要搜索一定数量（包括 0)的结构T (S 1), T (S 2), ..., T (S ⊆ S。由于A需要n Log r支付Ω(|S scan|)的成本来扫描S以获得一些

′ ′ ′) 并扫描子集S ，则|S scan| ≤ ′

′ ′ 扫描， α · 常数α > 0。我们将考虑r，我们知道它 r

是ω(1)，足够大以使

一个 · $\frac{n对数}{r}$ ⩽ n/2。因为 A 必须服从（5），所以我们可以断言

$$|S'_1 \cup 小号'_{2吨} \cup \ldots \cup S'| \geqslant |S| - |S'_{扫描}| \geqslant n/2。\quad (6)$$

这意味着t ⩾ 1。由于算法A必须花费常数时间来搜索每个结构（秒 $'_我$）(i ∈ [t])，则我们必须有

$$t = O((n/r) \cdot \mathrm{Log}\ r)。\quad (7)$$

然而，通过我们如何设计T ，建造T的总成本（S $'_1$), T (S $'_2$), ..., T (S $'_吨$）是

$$\sum_{我\in[t]} |S'_我| \cdot B(|S'_我|)。\quad (8)$$

设λ = $\sum_{我\in[t]} |S'_我|$;从(6)式可知λ ⩾ n /2。由于n · B(n)为凸函数，且B (n)，是非递减的，我们知道当|S $'_我$| = λ/t且对于所有i ∈ [t]。因此：

$$(8) \geqslant \lambda \cdot B(\lambda/t) \geqslant \frac{n}{2} \cdot B\ \frac{n}{2吨}。\quad (9)$$

从(7) 式可得n/(2t) = Ω(r/ log r)。因为B(n) = ω(log n)，所以基本渐近分析表明B(n/(2t))必定是ω(log r)。

我们现在得出结论，（9），因此（8），必须是ω（n Log r）。但这与算法 A 可以在O(n Log r) 时间内处理r 个查询。因此，我们假设 r = ω(1) 必定是错误的。

个DDS 算法， B(n) = ω(log n)。具有ω(n log n) 构造的数据结构时间对于 DDS 仍然有用，只要我们不沉迷于回答r 个查询 O(n Log r) 时间。为了使其正式化，我们修改定理 1 背后的技术以获得另一个具有以下保证的通用减少。

▶定理3.假设B(n)和Q(n)是非减函数，并且都是 O(logγ n)其中γ ⩾ 1为常数。每个(B(n), Q(n))谱可索引问题都承认时间（n， r）= O（n Logγ r)，其中 r ⩽ n。

证明与第 3.2 节中介绍的类似，并移至附录 A。

上述内容的一个有趣应用是R中的正交范围计数/报告 d 其中d是固定常数至少为 3（参见 1.2 节中的问题定义）。范围树增强具有分数级联 [9]，可在O(n logd−1 n) n点时间，回答 a 计数/报告查询同样在O(n logd−1 n) 时间。该问题可分解，因此(Logd−1 n, Logd−1 n) 频谱可索引。定理3直接给出了DDS算法时间（n， r）= O（n Logd−1 r)，对所有r ⩽ n，这严格改进了 [19] 的结果，即在第 1.2 节中提到。

参考

1 Alok Aggarwal 和 Prabhakar Raghavan。最近邻的延迟数据结构问题。信息处理快报（IPL）， 40（3）:119-122，1991。

2杰里米·巴尔贝、安库尔·古普塔、斯里尼瓦萨·拉奥·萨蒂和乔纳森·索伦森。接近最优内部和外部存储器中的在线多选。J .离散算法， 36:3-17，2016年。

3 Jon Louis Bentley 和 Hermann A. Maurer。范围的有效最坏情况数据结构搜索。Acta Inf.， 13:155–168，1980。

4 Manuel Blum、Robert W. Floyd、Vaughan R. Pratt、Ronald L. Rivest 和 Robert Endre Tarjan。选择的时间界限。计算机与系统科学杂志 (JCSS)， 7(4):448–461, 1973.

5格思·斯托廷·布罗达尔 (Gerth Stolting Brodal)、比特·格费勒 (Beat Gfeller)、艾伦·格隆伦德·乔根森 (Allan Gronlund Jorgensen) 和彼得·桑德斯 (Peter Sanders)。向最优范围中值。理论计算机科学， 412(24):2588–2601，2011。

6 Timothy M. Chan 和 Konstantinos Tsakalidis。二维和三维空间的最优确定性算法 3-d浅岩层。离散与计算几何， 56(4):866–881，2016年。

7 Bernard Chazelle、Leonidas J. Guibas 和 DT Lee。几何对偶的力量。BIT 数值数学， 25（1）：76-90，1985。

8 Yu-Tai Ching、Kurt Mehlhorn 和 Michiel HM Smid. 动态延迟数据结构。信息处理快报（IPL）， 35（1）：37-40，1990。

9马克·德伯格、奥特弗里德·张、马克·范·克雷维尔德和马克·奥维马斯。计算型几何：算法与应用。Springer -Verlag，第 3 版，2008 年。

10 Greg N. Frederickson 和 Donald B. Johnson。x+y 和带排序列的矩阵。计算机与系统科学杂志 (JCSS)， 24(2):197–208, 1982.

11 Beat Gfeller 和 Peter Sanders。迈向最优范围中值。国际自动机、语言和编程讨论会 (ICALP)，第 475–486 页，2009 年。

12 Goetz Graefe、Felix Halim、Stratos Idreos、Harumi A. Kuno 和 Stefan Manegold。并发性自适应索引控制。VLDB Endowment 论文集 (PVLDB)， 5(7):656–667， 2012 年。

13 Felix Halim、Stratos Idreos、Panagiotis Karras 和 Roland HC Yap。随机数据库破解：迈向主内存列存储中的稳健自适应索引。VLDB捐赠基金（PVLDB）， 5（6）：502–513，2012。

14 Sariel Har-Peled 和 Nirman Kumar。低维数据的近似最近邻搜索查询。SIAM计算杂志， 42(1):138–159，2013 年。

15 Sariel Har-Peled 和 S. Muthukrishnan。范围中值。欧洲研讨会论文集算法论（ESA），第 503-514 页，2008 年。

16 Stratos Idreos、Martin L. Kersten 和 Stefan Manegold。数据库破解。第 68–78 页， 2007 年。

17 Stratos Idreos、Martin L. Kersten 和 Stefan Manegold。自组织元组重建在列存储中。在ACM 数据管理 (SIGMOD) 论文集，第 297-308 页， 2009年。

18 Stratos Idreos、Stefan Manegold、Harumi A. Kuno 和 Goetz Graefe。合并已破解的内容，破解合并的内容：主内存列存储中的自适应索引。VLDB捐赠基金（PVLDB）， 4（9）：585–597，2011。

19 Richard M. Karp、Rajeev Motwani 和 Prabhakar Raghavan。延迟数据结构。SIAM 计算杂志, 17(5):883–902, 1988。

20 David G. Kirkpatrick。平面细分中的最佳搜索。SIAM计算杂志， 12(1):28–35, 1983.

21 David G. Kirkpatrick 和 Raimund Seidel。终极平面凸包算法？ SIAM 计算机杂志， 15（1）：287-299，1986。

22 Robert Krauthgamer 和 James R. Lee。导航网络：邻近度的简单算法搜索。在ACM-SIAM 离散算法年度研讨会 (SODA) 论文集上， 第798–807页，2004年。

23康斯坦丁诺斯·兰普罗普洛斯、法特梅·扎尔巴尼、尼科斯·马穆利斯和帕纳吉奥蒂斯·卡拉斯。高维度量空间中的自适应索引。VLDB 捐赠基金论文集（PVLDB）， 16（10）：2525–2537，2023年。

24 Rajeev Motwani 和 Prabhakar Raghavan。延迟数据结构：查询驱动的几何搜索问题预处理。在计算科学研讨会论文集几何（SoCG），第 303-312 页，1986 年。

25 Mark H. Overmars 和 Jan van Leeuwen。飞机配置的维护。期刊计算机与系统科学学报(JCSS), 23(2):166-204, 1981。

26 Bryce Sandlund 和 Sebastian Wild。惰性搜索树。在IEEE 年度论文集上计算机科学基础研讨会 (FOCS)，第 704–715 页，2020 年。

27 Bryce Sandlund 和 Lingyi Zhang。可选堆和最佳惰性搜索树。在ACM-SIAM 离散算法年度研讨会 (SODA) 论文集,第 1962-1975页,2022 年。

28 Neil Sarnak 和 Robert Endre Tarjan. 使用持久搜索树进行平面点定位。ACM通讯(CACM), 29(7):669–679,1986。

29菲利克斯·马丁·舒克内希特、阿莱克·金达尔和延斯·迪特里希。实验评估和数据库破解分析。 VLDB杂志, 25(1):27-52,2016年。

30 Yufei Tao 和 Dimitris Papadias。空间数据库中的范围聚合处理。IEEE知识与数据工程学报 (TKDE), 16(12):1555–1570,2004 年。

31 Fatemeh Zardbani、Peyman Afshani 和 Panagiotis Karras。重温数据库破解的理论和实践。在《扩展数据库技术 (EDBT) 论文集》中,第 415–418 页, 2020 年。

32 Fatemeh Zardbani、Nikos Mamoulis、Stratos Idreos 和 Panagiotis Karras。具有空间范围的对象的自适应索引。VLDB Endowment (PVLDB) 论文集, 16(9):2248– 2260,2023 年。

33 Donghui Zhang、Vassilis J. Tsotras 和 Dimitrios Gunopulos。对具有范围的对象进行高效聚合。在ACM 数据库系统原理研讨会(PODS) 论文集,第 121-132 页,2002 年。

定理 3 的证明

我们的算法以epoch 为单位执行。在第 i 个(i ⩾ 1) epoch开始时,我们设置s = 22 716 ,并在O(n · B(s)) = O(n · 2 i · γ) 时间内在S上创建一个结构T (即(B(n), Q(n))频谱可索引性所承诺的结构) 。该结构允许我们在O(n · Q(s))时间内回答任何查询。当s 个查询在该 epoch期间得到回答后,第 i 个 epoch 结束。

s

这些查询的总成本为

仅 $\frac{n}{s}$ · Q(s) = O(n · Q(s)) = O(n ·2i ·γ)。

因此,第 i 个时期的总计算时间为O(n · 2 i · γ)。因为γ⩾1 ,我们知道当i增加1时, n · 2i ·γ至少翻倍。因此,所有时期的总成本为渐近受O(n·2支配, h 的值是满足两个条件的最小整数 ·γ),其中h 是所需的 epoch 数。确切地说,

h ⩾ 1:⩽ n (s的值不能超过n);

C1: 2

C2: 我=1 2 2 ⩾ r (h 个时期可回答的查询数量必须至少为r)。

令H表示最小整数h ⩾ 1,使得 2 ⩾ r。这意味着

2 2 H−1 2 < r ⇔ 2 < r2 。 (10)

当 2 ⩽ n,上述论证保证了迭代次数h 最多为H (不等式 2 个时期的总成 ⩽ n必然有 2 ⩽ n,满足条件C1) 。此时所有·γ) = O(n · 2 H·γ) = O(n Logγ r)。本为O(n · 2

如果 2 > n,上述论证就不成立。然而,当这种情况发生时,我们从(10)知道r > n,从而导致r > √ n。一旦r达到 √ n 即捕捉点 我们就会在O(n logγ n) 时间内在整个S上创建一个结构T ,并使用它回答O(Q(n)) = O(logγ n) 时间内的每个后续查询,直到r = n。捕捉点之后的查询总成本为O(n logγ n) = O(n logγ r)。因此,我们得到了一个算法,该算法保证对于所有r ⩽ n, Time(n, r) = O(n Logγ r),从而完成了定理 3的证明。