DeepL
订阅DeepL Pro以翻译大型文件。
欲了解更多信息，请访问www.DeepL.com/pro。

最大限度地优化延迟的优化链数据结构（又称数据库破解）

摘要

本文研究如何最大限度地降低对一个数据库中 n 个元素进行 r 次查询的总成本。在线方式（即只有在前一个查询结果准备就绪后才进行下一个查询），当值 $r \le n$ 事先未知。传统的索引法首先要在数据上建立一个完整的索引。n 个元素，然后再回答查询，这样做可能不合适，因为索引的构建时间通常为 Ω(n log) - 可能成为性能瓶颈。与此相反，对于许多问题，a (n log(1+))的下限对每 ∈ [1,] 的 次查询的总成本成立。匹配这一下限是*延迟数据结构化* (DDS) 的主要目标，DDS 也称为*数据库系统界的破解*。针对各类问题，我们提出了通用的还原方法将传统索引转换为与长范围下限相匹配的 DDS 算法的 r。对于一个可分解的问题，如果一个数据结构可以在 $O(n \log n)$ 时间内建立，并且具有 $Q(n)$查询搜索时间，我们缩减后得到的算法在 $O(n \log(1+r))$时间内运行所有 $r \le \frac{n) (\log) (n)}{Q(n)}$，其中上界 $\frac{n \log (n)}{Q(n)}$是在温和约束条件下渐进的最佳值。特别是，如果 $Q(n) = O(\log n)$，那么 $O(n \log(1+r))$时间保证扩展到所有 $r \le n$，其中，我们以最佳解决了大量的 DDS 问题。我们的结果可以推广到是一类 "频谱可索引问题"，它包含可分解问题。

2012 ACM 学科分类 计算理论→ 数据结构与计算算法
数据管理

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...

导言

传统索引首先在整个数据集上创建索引，然后才开始回答问题。查询。，在一个（未排序的）S 集上建立一棵二叉搜索树（BST）之后 n 个实值的时间为 $O(n \log n)$，我们可以使用树在以下时间内回答每个前置查询[1] O(log n) 时间。然而，当数据集 S 只被搜索时，这种模式就会出现缺陷次。在极端情况下，如果只需要回答一个查询，那么最好的方法是方法是全面扫描 S，这只需要 $O(n)$ 时间。更一般地说，如果我们要以*在线*方式回答查询，即在查询结果出来后才给出下一个查询。上一次查询已经输出--总成本可能为 $O(n \log(1+r))$[19]。当 $r \ll n$ 时（例如，$r = \text{polylog } n$ 或 $2^{\sqrt{\log n}}$)时，成本 $O(n \log(1+r))$打破了 Ω(n log)，说明排序不必要的。

在许多大数据应用中都会出现类似上述情况，在这些应用中，大量数据收集，但很少查询。这一现象推动了一系列研究，系统界称之为*数据库破解*；见 [12, 13, 16-18, 23, 29, 31, 32] 及其参考文献。在这些环境中，问题的规模 n 非常大，甚至可以达到排序被认为是昂贵的，应。此外不可能预测会有多少查询，即前面提到的整数 r。需要支持 。与立即构建完整的索引相比，数据库破解在每次查询中只进行计算部分的构造。如果 r 最终

[1] 给定搜索值 q，前置查询返回 S 中不超过 q 的最大元素。


©;
采用知识共享许可协议 CC-BY 4.0 许可 莱布尼茨国际信息学会议

录
达格施图尔城堡 - 莱布尼茨信息中心，达格施图尔出版社，德国

达到某个阈值时，将创建索引，但 r
会在某个明显低于该阈值的值时停止。数据库的挑战
破解是为了确保 "平滑性"：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，数据库破解的核心是数据结构问题。
1988 年，Karp、Motwani 和 Raghavan [19]已经以*延迟为名*进行了研究。
数据结构（DDS）。对于一些问题，他们解释了如何回答 $r \in [1, n]$
对 n *个*元素进行查询，总成本为 $O(n \log(1+r))$，事先不知道 r。对于
每 $r \in [1, n]$，他们证明了这些问题的匹配下限为(n log(1+))。
通过还原，同样的下限在许多其他 DDS 上也被证明是成立的。
问题。

这项工作探讨了以下主题：如何设计一种通用还原法，在给定的条件下
一个（传统）数据结构，能否*自动*转化为高效的 DDS 算法？
这种减少可能会大大简化 DDS 算法的设计，并带来新的启示。
传统索引和 DDS 之间错综复杂的联系。

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

1.1 问题定义

本小节将正式提出要研究的问题。设 S 称为*数据集*、
是一个从域 D 中抽取 n *个*元素的集合。让 Q 是一个（可能是无限的）集合，其中
这些元素称为*谓词*。给定一个谓词 $q \in Q$，在 S 上发出的*查询*会返回一个
答案，表示为 $\mathrm{ANS}_q(S)$。我们认为，对于任意 $q \in Q$，答案 $\mathrm{ANS}_{(q)}(S)$ 可以
可以用 $O(1)$ 个字表示，并且可以在 $O(n)$ 时间内计算（这基本上意味着
，查询可以用穷举扫描等暴力算法来回答）。

推迟数据结构（DDS）。我们现在正式提出 DDS 问题。最初，一个
算法 A 数据集 S 是一个数组，其中的元素是任意的
permuted。对手首先为第一个查询选择一个谓词 q_1，然后征求
答案 答案$_{q_1}(S)$来自 A .对每个 $i \geq 2$，在第 (i) − 第 1 个查询回答
后
在获得 之后，对手要么决定终止整个过程，要么选择
下一次查询的谓词 $q_{(i)}$，并向 A 请求 $\mathrm{ANS}_q(_{i)}S)$。允许对手
观察 A 的执行情况，从而能够为 A 选择一个 "坏 "q_i。

对于 t *个*查询，算法 A 可以保证*运行时间* $\mathrm{TIME}(n,r)$，如果对于*每个*
$r \leq t$，处理前 r *个*查询的总成本最多为 $\mathrm{TIME}(n, r)$。 我们将
专注于 $t \leq n$，因为这是数据库破解的重要场景。

最佳状态。虽然上述设置是 DDS 的 "标准"，但就数据库而言
就裂缝问题而言，从另一个全新的角度进行调查是有意义的。
由于数据库破解主要是在数据集接收相对较多的数据的情况下使用。
查询次数很少，因此必须设计在下列情况下特别高效的 DDS 算法
查询次数很少。

为了使 "特别有效 "的概念正规化，我们可以利用这样一个事实，即对于许多.....

DDS 问题（包括本工作中考虑的问题），存在硬度障碍

在相关计算模型下，对于每∈ [1,]，TIME(n,*r*)= Ω(n Log)。

受此启发，我们说算法 A 保证了 LOGSTREAK(*n*) 的 log(*r*)-STREAK

如果对于所有 *r*≤ LOGSTREAK(*n*)，其运行时间满足 TIME(*n, r*)= *O*(*n* LOG *r*)。

最差的 log(*r*)-streak 保证是 LOGSTREAK(*n*) = *O*(1) - 这是微不足道的，因为任何查询都能在 *O*(*n*) 时间内得到回答。理想情况下，我们希望 LOGSTREAK(*n*)= *n*，但正如本文后文所述，这并不总是可能的。不过，在实际应用中，算法只要能确保在某个常数ϵ> 0 时 LOGSTREAK(n) = Ω(n(ϵ)) 就足够了因为(n$^{\epsilon}$查询对于数据库破解来说可能已经太多，没有吸引力。

问题类别。上述定义框架可以专门划分为各种问题实例，它们在数据域 D、谓词域 Q 和结构上各不相同

接下来，我们介绍与我们的讨论相关的两类问题：

- 如果对于任意不相交的集合 S_1、S_2 D 和任意谓词 *q* Q，可以在恒定时间内从 ANS$_{(q)}$(S_1) 和 ANS$_{(q)}$($S_{(2)}$)推导出 ANS($_q$)($S_{(1)}$ S$_{(2)}$)，则问题实例是*可分解的*。
- 对于每个数据集 *S* *D*，如果问题实例具有以下属性，我们就将其定义为（*B*(*n*), *Q*(*n*)）*频谱可索引*问题实例：对于每个整数 *s* [*S*]，都可以在 *O*(B(s^{l}) 内在 *S* 上构建一个数据结构，并在 *O*($^{\perp s\perp}$ *Q*(s^{l})时间内回答任何查询。选择 "频谱可索引 "一词，是为了反映为参数 *s* 的整个频谱建立 "良好 "索引结构的能力--就函数 *B*(*n*) 和 *Q*(*n*) 而言。

关于上述定义，有两点很重要：

- (*B*(*n*), *Q*(*n*)) *频谱可索引性*意味着我们可以在 *O*(| *S* |- *B*(| *S*)) |时间内在任意数据集 *S*⊆ D 上构建一个数据结构，在 *O*(*Q*(S)) ||时间内回答查询（为此，只需设置 *s* *S* ）= ||。
- 考虑任何具有以下性质的可分解问题实例：对于任何数据集 *S* D，我们都可以在 *O*(*S* B(*S*)) 时间内建立一个数据结构 ，从而在 *O*(*Q*(*S*)) 时间内回答查询。那么，问题实例必须是 (*B*(*n*), *Q*(*n*)) 谱可索引的。为了说明原因，给定一个整数 *s* [*S*]，将 *S* 任意划分为 *m*= *S* /*s* 不相交的子集 S_1、S_2、…..、$S_{(m)}$，其中除了 S_m，所有子集的大小都是 *s*。对于每个 *i* [*m*]，在 *O*($S_{(i)}$ $_{B(s)}$)时间内创建一个结构 (*S*(*i*))；创建所有 *m* 个结构的总时间为 *O*(*m* *s* B(*s*))= O(*S* B(*s*))。要回答查询 *q*，只需搜索每个 (*S*$_i$)，即可在 *O*(*Q*(*s*))) 时间内获得 ANS$_{(q)}$($S_{(i)}$))，然后用 *O*(*m*) 时间将 ANS($_{q}$)(S_{1})、ANS$_{(q)}$(S_2)、…、ANS$_{(q)}$($S_{(m)}$))合并为 ANS($_q$)($S_{}$)。因此，总查询时间为 *O*(*m* - *Q*(*s*))。

1.2 相关工作

莫特瓦尼和拉加万在一篇会议论文[24]中介绍了 DDS 的概念，该论文被合并到与卡普共同撰写的期刊文章[19]中。他们 [19] 为以下 DDS 问题设计了算法：

- *前置词搜索*，其中 *S* 由 *n* 个实值组成，每个查询给定一个任意值 *q*，并返回 *q* 在 *S* 中的前置词。
- *半平面包含*，其中 *S* 由 R^2 中的 *n* 个半平面组成，每次查询给出一个任意点 *q*∈ R(2)，并返回 *q* 是否被 *S* 中的所有半平面覆盖。
- *凸壳包含*，其中 *S* 由 R^2 中的 *n* 个点组成，每次查询给出一个任意点 *q*∈ R(2)，并返回 *q* 是否被 *S* 的凸壳覆盖。
- *二维线性编程*，其中 *S* 由 R^{2}中的 *n* 个半平面组成，每次查询都会给定一个二维向量 ***u***，并返回所有 *n* 个半平面交点中使 ***u*** - ***p*** 的点积最大的点 *p*。
- *正交范围计数*，其中 *S* 由 R (d) 中的 *n* 个点组成，维度为 *d* 是一个固定常数，而查询则是给定一个任意 d- *矩形 q*，即一个

轴平行方框的形式 $[x_1, y_1] [x_2, y_2] \ldots [x_d, y_d]$ - 并返回 q 所覆盖的 S 点的个数。

对于前四个问题，Karp、Motwani 和 Raghavan 提出了对所有 r n 实现 $\text{TIME}(n, r) = O(n \text{ Log } r)$ 的算法。对于正交范围计数，他们提出了两种算法，第一种算法确保对所有 r N 的 $\text{TIME}(n, r) = O(n \text{ Log}^d r)$，而另一种算法确保对所有 r n 的 $\text{TIME}(n, r) = O(n \log n + n \text{ Log}^{(d)-1} r)$。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，对于每一个 [1,]，任何算法的运行时间必须满足 $\text{TIME}(n, r) = \Omega(n \text{ Log })$。

Aggarwal 和 Raghavan [1] 提出了一种 DDS 算法，对于所有 r n，*近邻搜索* $\text{TIME}(n, r) = O(n \text{ Log } r)$，其中 S 由 R^2 中的 n 个点组成，查询给定 R^2 中的任意点 q，并返回 S 中与 q 最接近的点。对于所有 r n，该运行时间在代数模型下都是最优的。

关于*范围中值的* DDS，我们可以讲一个 "成功的故事"。对于任意 1 x y n，让 $A[x : y]$ 表示元素 $A[x]$、$A[x+1]$、...、$A[y]$ 的集合。此外，我们还给出了 r 个整数对 $(x_1, y_1), (x_2, y_2), \ldots, (x_r, y_{(r)})$，使得 1 $x_{(i)}$ $y_{(i)}$ n，对于每个 i [n]。我们的目标是找出所有 i [r] 的集合 $A[x_i : y_{(i)}]$ 的中值。在 [15] ，中Har-Peled 和 Muthukrishnan 解释了如何在 $O(n \text{ Log } r + r \log n \text{ Log } r)$ 时间内解决这个问题，并证明了在所有 n 的比较模型下，(n Log) 的下限。在 [11] 中（另见 [5]），Gfeller 和 Sanders 考虑了问题的 DDS 版本，其中 S 如前定义，查询给定任意一对 (x, y)，1 x y n，返回 $A[x : y]$ 的中值。他们设计了一种算法，在所有 的情况下，$\text{TIME}(n, r) = O(n \text{ Log } r)$。不难看出，任何 DDS 算法都可以在 $\text{TIME}(n, r)$ 时间内解决离线问题。因此，Gfeller 和 Sanders 的算法改进了 [15] 的离线解法，并且在比较模型下（对 DDS 而言）是最优的。

Ching, Mehlhorn 和 Smid 考虑了一个动态 DDS 问题，在这个问题中，除了查询之外，还需要一种算法来支持数据集 S 的更新。这个方向也被扩展到了动态 DDS；请参见近期的著作[26, 27]。

如前所述，系统界以数据库破解为名对 DDS 进行了广泛研究。其重点是设计高效的启发式方法，以加速各种实际场景中遇到的查询工作量，而不是建立强有力的理论保证。有兴趣的读者可以参考 [12, 13, 16- 18, 23, 31, 32] 等文献中的代表作，作为进入该领域的切入点。

1.3 我们的成果

我们的主要结果是一个具有以下保证的通用还原：

► 定理 **1.** *假设 $B(n)$ 和 $Q(n)$ 是非递减函数，使得 $B(n) = O(\log n)$ 和 $Q(n) = O(n^{1-\epsilon})$ 其中 $\epsilon > 0$ 是一个任意小的常数。每个（$B(n), Q(n)$）频谱可索引问题都有一个 DDS 算法，其运算法则为*

$$\text{LOGSTREAK}(n = \min\left(n, \frac{c(-)(n)(\log n)}{Q(n)}\right)(,) \quad (1)$$

对于一个任意大的常数 c，即算法达到 $\text{TIME}(n, r) = O(n \text{ - Log } r)$

对于所有 $r \leq \min\{n, \frac{c_1(n)(\log n)}{B(n)}\}$。

正如我们在第 3 。节中所论证的，在温和的约束条件下，(1) 中的链式约束是可能的最佳约束一个重要的是，当 $Q(n)= O(\log n)$ 时，我们的还原法产生的 DDS 算法实现了 LOGSTREAK$(n)= n$，即对于所有 $r \le n$，TIME$(n, r)= O(n \text{ Log } r)$。对于许多对数据库系统非常重要的可分解问题，具有 $O(n \log n)$ 构造时间和 $O(\log n)$ 搜索时间的数据结构已经为人所知（例如，前置搜索和最近邻搜索）。由于这些问题必须是（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 版本提供了出色的解决方案，这些解决方案通常在比较/代数模型下最优的。第 4.1 。节将给出这些问题的精选

定理 1 给数据库破解带来了一个令人愉快的启示：即使是查询时间较慢的数据结构也能用于破解！例如，对于正交范围计数（定义为

根据第 1.2 节中在二维空间中的计算结果，kd 树的构建时间为 $O(n \log n)$，其回答查询的时间为 $Q(n)= O(\sqrt{n})$。定理 1 表明，利用该结构可以在 $O(n \text{ Log } r)$ 时间内回答任意 $r= O(\sqrt{n} \log n)$ 查询，这对于我们来说可能绰绰有余。
现实中的数据库破解。这很好地体现了我们（在第 1.1 ）节中的动机，即
研究 "DDS 算法对小 r 特别有效"。

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的*可索引谱*到底*多大*。正如我们将在第 4.2 节中演示的半平面包含、凸壳包含、范围中值和二维线性规划（它们的定义见第 1.2 节），在这个方向上的探索会相当有趣。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, $Q(n)$）谱可索引的，然后利用定理 1 获得一种算法，对于所有 $r \le n$，其求解时间为 TIME$(n, r) = O(n \text{ Log } r)$。

那么，需要$\omega(n \log n)$时间来构建的结构，或者等价于 $B(n) = \omega(\log n)$ 的结构会怎么样呢？第 5 节将说明，在温和的约束条件下，任何通用还原法都无法利用这种结构获得 LOGSTREAK$(n) = \omega(1)$ 的 DDS 算法。换句话说，只有在 $r= O(1)$ 的微不足道的情况下，这些算法才能实现 TIME$(n, r)= O(n \text{ Log } r)$。尽管如此，如果我们接受 "对于所有 $r \le n$，TIME$(n, r) \le n \text{ Log}^{O(1)} r$ "这种形式的保证，那么只要 $B(n)$ 和 $Q(n)$ 是 $\text{Log}^{O(1)} n$，我们的定理 1 的还原就可以扩展到生成这样的，这将在第 5 节中讨论。

2 ：前身搜索

前置词搜索是 DDS 和数据库破解领域最受关注的问题。本节将回顾 [19] 中开发的两种方法，这两种方法可在该问题上实现 TIME$(n, r)= O(n \text{ Log } r)$。这些方法构成了几乎所有现有 DDS 算法的基础。

自下而上回顾一下，数据集 S 由 n *个*实值组成。我们假定 n 是 2 的幂次。在任何时候，数据集 S 都会被任意分割成不相连的子集（称为*运行*），对于某个 $i \ge 0$，每个子集都具有相同的大小 $s= 2^{(i)}$。每个运行都会被排序并存储在一个数组中。最初，$s= 1$，即一个运行包含 S 的一个单独元素。随着时间的推移，运行大小 s 会单调增加。每当某个值 $j > i$ *的* s 需要从 2^i 变为 2^j 时，就会进行一次*大修*，以建立新的运行。由于合并 2^{j-i} 大小为 $2^{(i)}$的运行可在 $O(2^j (j - i))$ 时间内获得大小为 $2^{(j)}$的运行，因此大修可在 $O(n (j - i))$ 时间内完成。因此，如果当前运行规模为 s，则生产历史上所有运行的总成本为 $O(n \text{ Log } s)$。

每次运行时，只需执行二进制搜索即可回答（前置）查询。不过，为了控制成本，算法会确保当前大小 s 至少为

i 处理第 i 个 (i) 查询之前的日志 i ≥ 1) 查询。如果不满足条件，则进行大修调用 将 s 增加到最接近的 2 的幂，至少为 $i \operatorname{Log} i$。，查询需要花费 $O(\frac{n}{\text{Log } s}) = O(\frac{n}{(i \operatorname{Log} i)}) = O(n/i)$ 时间。如果我们把这些加起来对于所有 r 个查询，总和变为 $O(n \cdot \sum_{i=1}^{r} \frac{1}{i}) = O(n \operatorname{Log} r)$。由于最终运行规模 s 为 $O(r \operatorname{Log} r)$，因此我们可以得出结论：该算法处理 r 个查询只需 $O(n \operatorname{Log} r)$ 时间。注意只有当 $r \operatorname{Log} r$ n（最大运行规模为 n）时，这一点才成立。然而，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 $r \leq n$，该算法都能实现 $\text{TIME}(n, r) = O(n \operatorname{Log} r)$。

自上而下。这种方法模仿了在 S 上构建二叉搜索树（BST） 的以下策略：(i) 找出 S 的中位数，并在中位数处将 S 分成 S_1 和 S_2；(ii) 将中位数存储为根的键，然后在 S_1（或 $S_{(2)}$）上递归地构建根的左（或右）子树。该算法在查询处理过程中 "按需 "构建 ，而不是直接进行完整的构建。

一开始，只有根节点存在，它处于*未展开*模式。一般来说，未展开节点 u 还没有子节点，但与子节点的子集 S_u S 相关联，该子集包含了应存储在其子树中的元素。回答查询的方式与普通 BST 相同，都是遍历 BST 的根到叶路径 π。主要区别在于当搜索到 π 上的未展开节点 u 时，算法必须先*展开*它。扩展 u 意味着为 u 创建两个子节点，在中位数（u 的关键点）处分割 S_u，将中位数处的 $S_{(u)}$ 分成两部分，并为每一部分关联一个子节点。之后，u 会被*扩展*，其子节点会被放入未扩展的模式中。扩展需要 $O(S_{u})$ 时间（找到集合的中位数需要线性时间 [4]）。

经过 r 次查询后，BST 将被部分构建，因为只有在查询处理过程中遍历的 r 条根到叶路径上的节点才会被扩展。第一个 Log r 层的节点[2] 的总扩展成本为 $O(n \operatorname{Log} r)$。对于每条根到叶路径 π，Log r 层的节点 u 的扩展成本为 $O(n/r)$，它支配着 π 上 u 的子节点的总扩展成本。因此，除了第一个 Log r 层的节点外，π 中所有其他节点的总扩展成本为 $r \cdot O(\frac{n}{r}) = O(n)$。因此，对于所有 $r \leq n$，该算法都能实现 $\text{TIME}(n, r) = O(n \operatorname{Log} r)$。

3 从数据结构到 DDS 算法的还原

第 3.1 节将把自下而上的方法（在上一节中进行了回顾）扩展为一种通用还原法，只要满足一个关键要求--线性可合并性（稍后定义）--它就能产生具有较大 log(r)-streak 边界的 DDS 算法。尽管这一还原将被我们在定理 1 但下面的最终还原（在第 3.2 ）节中介绍所取代，对它的讨论i）加深了读者对这一方法的威力和局限性的理解，（ii）阐明了在缺乏线性可合并性的情况下新思想的必要性。最后，第 3.3 节将建立一个硬度结果，以说明定理 1 中的条纹约束不再能得到显著改进。

3.1 第一次还原：推广自下而上的方法

本小节将重点讨论可分解问题。对于任何数据集 $S \subseteq D$，我们假设存在一种数据结构 $\mathcal{D}(S)$，它能在 $O(Q(n))$ 时间内回答任何查询。此外，我们将重点讨论可分解问题、

[2] 节点的*级别*是指从节点到根的路径上的边数。

结构是*线性可合并的*，即对于任意不相交的 S_1、$S_2 \subseteq D$，$S_{(1)} \cup S_{(2)}$ 上的结构可以在 $O(|S_{(1)}|+|S_{(2)}|)$ 时间内由 $\mathscr{A}(S_{(1)})$ 和 $\mathscr{A}(S_{(2)})$ 构建。请注意，这意味着 $\mathscr{A}(S)$ 可以在 $O(n \log n)$ 时间内构建。

► 定理 **2.** *对于存在线性可合并结构的可分解问题，其查询时间 $Q(n)=O(n^{1-\epsilon})$ 其中 $\epsilon > 0$ 是一个常数，我们可以设计一种 DDS 算法，保证在任意大的常数 c 下，* $\textsc{LogStreak}(n)=\min_{Q(n)}\{n, c(1-\epsilon)n^{\epsilon}\log^{\epsilon}(n)\}$。

证明我们假定 n 是 2 的幂次。与自下而上的方法一样，在任何时刻，我们都可以任意地将 S 分割成大小相同的运行 $s=2^i$ 对于某个 $i \geq 0$。对于每个运行，在其中的元素上建立一个结构。初始运行大小 s 为 1。某个值 $j > i$，每次 s 从 2^i 增长到 $2^{(i)}$，都要进行一次*大修*，以构建大小为 2^j 的运行。根据线性可合并性，我们可以在 $O(2^j - 2^i)$ 时间内，通过合并大小为 $2^{(i)}$ 的 $2^{(j)-i}$ 运行的结构来构建大小为 $2^{(j)}$ 的运行结构。通过与第 2，节类似的分析如果当前运行大小为 s，则为历史上出现过的所有运行生成结构的总成本为 $O(n \operatorname{Log} s)$。

要回答谓词 q 的查询，需要搜索每次运行的结构，然后所有运行的答案合并为 $\textsc{Ans}_q(S)$。查询成本为 $O(\frac{n}{s} \cdot Q(s))$。我们要求，在回答*第 i 个*（$i \geq 1$）查询之前，运行规模 s 必须满足以下条件

$$Q(s)/s \leq 1/i。 \tag{2}$$

如果达不到这一要求，我们就会启动大修，将 s 增加到满足 (2) 的 2 的最小幂。这样就能确保以 $O(n/i)$ 的成本回答*第 i 次*查询。因此，处理 r 个查询的总成本为 $O(n \operatorname{Log} r)$。

剩下的工作就是确定大修的成本。由于 (2)，最终运行规模 s 是满足以下条件的 2 的最小幂

- $s \leq n$（运行规模不能超过 n）和
- $s/Q(s) \geq r$（因为 (2)）。

由于 $Q(s) = O(s^{1-\epsilon})$，我们知道 $Q(s) \leq \alpha s^{1-\epsilon}$，对于某个常数 $\alpha > 0$。因此，$s/Q(s) \geq s^{\epsilon}/\alpha$。由于我们的目标是找到 s 的上界，我们要求 s 满足 $s^{\epsilon}/\alpha \geq r$（这比 $s/Q(s) \geq r$），或等价于 $s \geq (\alpha r)^{1/\epsilon}$。因此，如果 $(\alpha r)^{1/\epsilon} \leq n$，那么我们可以声称 $s = O(r^{1/\epsilon})$，因此 $O(\operatorname{Log} s) = O(\operatorname{Log} r)$，在这种情况下，所有检修总体需要 $O(n \operatorname{Log} r)$ 时间。

如果 $(\alpha r)^{1/\epsilon} > n$，则上述策略不起作用。不过，在这种情况下，$r > n^{\epsilon}/\alpha$，也就是说，$r$ 已经是 n 的多项式了。这就激发了下面的暴力策略。当 r 达到 n^{ϵ}/α 时--我们称这一时刻为*抢断点*--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，并用它在 $O(Q(n))$ 时间内回答随后的所有查询，直到 $r = \min_{Q(n)}\{n, c(1-\epsilon)n^{\epsilon}\log^{\epsilon}(n)\}$。抢答点之后的所有查询的总费用为 $O(n \log n) = O(n \log r)$。因此，我们得到了一种算法，对于所有 $r \leq \min_{Q(n)}\{n, c(1-\epsilon)n^{\epsilon}\log^{\epsilon}(n)\}$，都能保证 $\textsc{Time}(n, r) = O(n \operatorname{Log} r)$。 ◄

上述缩减的关键在于结构是线性可合并的。否则，创建大小为 2^i 的运行时的大修费用将达到 $(n \log(2^{(i)}))$，在这种情况下，所有大修的总费用将达到 $\Omega(n \operatorname{Log}(2^r))$。接下来，我们将介绍另一种可减少 $\operatorname{Log} r$ *倍*的缩减。

3.2 第二次还原：无线性合并

现在我们将放弃第 3.1 节中的线性可合并性要求，并建立定理 1。回顾一下，底层问题是 $(B(n), Q(n))$ 谱可索引的，$B(n) = O(\log n)$

和 $Q(n)= O(n^{1-\epsilon})$，对于某个常数 $\epsilon > 0$。我们的目标是设计出一种算法，它能够

$\text{TIME}(n, r)= O(n \text{ Log } r)$ for all $r \leq \min\{n, c \cdot n \cdot \log n\}$，其中 $c > 0$ 可以是任意常数。

假定 n 是 2 的幂次。我们的算法以*时间*为单位执行。在*第 i 个*（$i \geq 1$）纪元开始时，我们设置

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

并在 $O(n - B(s))$ 时间*内在 S* 上创建一个结构 $\mathcal{D}$，即 $(B(n), Q(n))$ 谱可索引性所承诺的结构。该结构允许我们在 $O(\frac{n}{s} \cdot Q(s))$ 时间内回答任何查询。

第 i *个*纪元在经过 $s/Q(s)$次查询后结束[3]要求的总 *在*该时间段*内*得到回复。这些查询

成本为

$$\frac{s}{Q(s)} \cdot O\left(\frac{n}{s} \cdot Q(s)\right) = O(n).$$

由上可知，第 i *个* epoch 的总计算时间为 $O(n-B(s))= O(n - 2^{(i)})$。当 i 增加 1 时，$n - 2^{(i)}$ 会加倍，因此回答 r 次查询的总成本为 $O(n \cdot 2^h)$，其中 h^* 是所需的历元数。准确地说，h^* 的值是满足两个条件的最小整数 $h \geq 1$：

- **C1:** $\sum_{i=1}^{h} 2^{2^i} \leq n$（$s$ 的值不得超过 n）；
- **C2:** $\sum_{i=1}^{h} \frac{2^{2^i}}{Q(2^{2^i})} \geq r$（$h$ 个纪元所能回答的查询次数必须是至少为r）。

由于我们的目标是找到 h 的上界，因此我们用一个更严格的条件 **C2** 代替条件：$2^{(2^{(h)}}/Q(2^{(2^{(h)}}) \geq r$.因为 $Q(n)= O(n^{1-\epsilon})$, 所以我们知道 $Q(n) \leq \alpha \cdot n^{(1)-\epsilon}$ for some 常量 $\alpha > 0$。，我们将 **C2** 进一步修改为更严格的不等式：

$$\frac{2^{2^h}}{\alpha \cdot (2^{2^h})^{1-\epsilon}} = \frac{(2^{2^h})^{\epsilon}}{\alpha} \geq r \quad \Leftrightarrow \quad 2^{2^h} \geq (\alpha \cdot r)^{1/\epsilon}. \tag{3}$$

让 H 代表验证 (3) 的最小整数 $h \geq 1$。这意味着

$$2^{(2)^{(H)-1}} < (\alpha \cdot r)^{(1)(/\epsilon)} \quad \Leftrightarrow \quad 2^{2^H} < (\alpha \cdot r)^{(2)/\epsilon}. \tag{4}$$

当$2^{(2)^H} \leq n$时，上面的论证保证我们的历元数 h^* 最多为 H（不等式 $2^{2^H} \leq n$ 意味着 $2^{2^h} \leq n$，服从条件 **C1**）。在这种情况下，所有每个历时的总成本为 $O(n \cdot 2^h) = O(n \cdot 2^H)$，根据 (4) 可得 。$O(n \text{ Log } r)$

如果 $2^{2^H} > n$，则上述论证不起作用。然而，当这种情况发生时，我们从 (4) 知道 $(\alpha \cdot r)^{(2)/\epsilon} > 2^{(2)^H} > n$，导致 $r > n^{(\epsilon/2)}/\alpha$。一旦 r 达到 $n^{\epsilon/2}/\alpha$--*抢断点*--我们就在 $O(n \log n)= O(n \log r)$ 时间内在整个 S 上创建一个结构 $\mathcal{D}$，并用它在 $O(Q(n))$ 时间内回答随后的每个查询，直到 $r = \min\{n, c \cdot n \cdot \log n\}$。抢断点之后的查询所需的总费用为 $O(n \log n)= O(n \log r)$。因此，我们得到了一种算法 $\mathcal{D}_{Q(n)}$，它能保证在所有 $r \leq \min\{n, c \cdot n \cdot \log n\}$ 时，$\text{TIME}(n, r) = O(n \text{ Log } r)$ 、

完成定理 1 的。证明

3.3 连贯约束的严密性

本节将解释为什么定理 中的条纹边界$(\frac{n \log n}{Q(n)})$是渐近的最适合具有 "合理 "行为的缩减算法。

[3] 在实践中，我们的算法可以通过使这个数字 $\lceil s - B(s)/Q(s) \rceil$ 略有改进，但这对于证明定理 1 无必要。

限制结构可分解问题的黑箱还原。为了证明硬度结果，我们可以随意*特化*问题类别，我们只考虑可分解问题。任何可分解问题，只要是（$B(n), Q(n)$）谱可索引的，都需要有一种还原算法。正如第 1.1 的，节所解释这意味着数据结构 $\mathscr{Y}$ 可以在 $O(S \cdot B(S))$ 时间内建立在任意 S 上，并在 $O(Q(S))$ 时间内回答 S 上的任意查询。

只要（$B(n), Q(n)$）频谱的可索引性得以保留，我们就可以限制$\mathscr{Y}$的功能，使其对 A 难以发挥作用。具体来说，$\mathscr{Y}$只为 A 提供以下 "服务"：

- A 可以在任何子集 $S' \subseteq S$ 上创建一个结构 $\mathscr{Y}(S')$；
- 给定谓词 $q \in Q$，A 可以使用 $\mathscr{Y}(S')$ 在 S' 上找到答案 $\mathrm{ANS}_{(q)}(S')$；
- 给定一个谓词 q，在 A 已经得到不相交子集 S_1'、S_2'，它可以在恒定时间内将答案合并为 $\mathrm{ANS}_{(q)}(S_1' \cup S_2')$（合并算法由 ）。

尽管其功能有限，但仍能使问题（$B(n), Q(n)$）谱可索引，因为问题是可分解的；参见第 1.1 节中的解释。

到目前为止，我们还没有对 A 的行为施加任何限制，但现在我们已经准备了。为了回答一个查询，算法 A 需要搜索 S 的若干子集（包括零）上的结构，例如，$\mathscr{Y}(S_1')$，$\mathscr{Y}(S_2')$，……，$\mathscr{Y}(S_t')$。算法可以选择任意 $t \geq 0$ 和任意 $S_1', S_2', \ldots, S_t'$（它们不需要是不相交的）。此外，还允许 A 通过支付 $\Omega(|S'_{can}|)$ 时间，检查另一个子集 $S_{(s)'can}$。综上所述，算法必须确保

$$S_{s'can} \cup S_1' \cup S_2' \cup \ldots \cup S_t' = S. \tag{5}$$

上述约束是自然的，因为否则 $S_{s'can} S_{1'} S_2' \ldots S_t' \ldots S_t'$.如果算法 无论如何都 "敢于 " 返回查询答案，那么它一定是获得了底层问题的某些特殊属性。在这项工作中，我们感兴趣的是不考虑特定问题属性的通用还原。

我们将把符合上述要求的还原算法称为*黑箱还原*。我们在定理 2 和定理 1 中的算法就属于黑箱算法。

定理 1 的严密性我们将证明，任何黑箱还原只能回答 $= \Omega(n^2)$ 对于任何函数 $Q(n)$：$\mathbb{N} \to \mathbb{N}^+$，，满足 $Q(n) = O(n)$，且对于任意常数 c $Q(n) = \Theta(Q(cn)) > 0$；是次相加的，即 $Q(x+y) \leq Q(x) + Q(y)$ 对任何 x、y 都成立。 ≥ 1.

这将证实定理 1 在黑箱类上的严密性。

我们将想出一个可分解的问题和一个相应的数据结构，这两个问题在现实中毫无意义，但在数学上却是合理的。数据集 S 由 n 个任意元素组成；给定任意谓词，对 S 的查询总是返回 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 "一个数据结构时，我们都会故意浪费 $S \log S$ 的时间，然后简单地在数组中输出 S 的任意排列。每当被要求 "回答 "一个查询时，我们会故意浪费 $Q(|S|)$ 的时间，然后返回 S。这个问题显然是可以分解的。

我们认为，任何黑箱还原算法 A 都需要$(Q(n))$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 $\mathscr{Y}(S_1'), \ldots, \mathscr{Y}(S_t')$ 对于某个 $t \geq 0$ 并扫描 $S_{s')(can}$。根据我们的结构设计，查询成本至少为

$$\Omega(|S_s'_{can}|) + \sum_{i \in [t]} (|S_{(i)(}|) \geq (|S(s)'_{can}|) + Q\left(\frac{1}{2} \sum_{i \in [t]} |S(i'|\right) \quad \text{(通过次可加性)}$$

$$=\Omega(|S_{(s)(r)(can)}|)+Q\sum_{i\in[t]}|S_{(i)(r)}| \quad \text{(by } Q(n)=O(n),\ Q(n)=\Theta(Q(cn)))$$

$$=\Omega(|S_{s(r)(can)}|+\sum_{i\in[t]}|S_{(i)(r)}| \quad \text{(by subadditivity)}$$

$$=\Omega(Q(n))。 \quad \text{(由 (5)}$$

根据 log(r)-streak 的定义，算法 A 必须在总代价为 $O(n \log r)$ 的范围内处理 $r=$ LogStreak(n) 查询，而总代价显然不能超过 $O(n \log n)$（记住 $r \le n$）。因此，A 只能处理 $O(\frac{n \log(n)}{Q(n)})$ 次查询。

4 种针对具体问题的新 DDS 算法

现在，我们利用定理 1 为具体的 DDS 问题开发算法，第 4.1 节重点讨论可分解问题，第 4.2 节重点讨论不可分解问题。

4.1 可分解问题的应用

如前所述，如果一个可分解问题的数据结构可以在 $O(n \log n)$ 时间内构建（即 $B(n) = O(\log n)$），并且支持在 $Q(n) = O(\log n)$ 时间内进行查询，那么定理 1 就可以直接得到一个 DDS 算法，对于所有 $r \le n$，其 Time(n, r)= $O(n$ Log $r)$。下面我们列举了对数据库系统具有重要意义的此类问题的部分清单，在这些问题中，以前没有已知的具有相同保证的算法。

- *二维正交范围计数。*问题定义见第 1.2 节。结构可以是一棵持久的 "聚合 "二叉搜索树（BST）[30]。
- *矩形上的正交范围计数。*数据集 S 是 R^2 中 n 个 2-rectangles（即轴平行的方框）的集合。给定一个任意的 2- 矩形 q，查询会返回 S 中与 q 相交的矩形数量。[33。这个问题可以简化为上一个问题（*点*上的正交范围计数）的四个查询]
- *点定位。*数据集 S 是由 n *条*线段定义的 R^2的平面细分，其中每条线段都与与其相关两个面的 id 相关联。给定一个任意点 $q \in R^{(2)}$，查询会返回包含 q 的细分面，简单来说就是找到紧靠 q 上面的线段（即从 q 向上射出的射线 "击中 "的第一个线段）。该结构可以是持久 BST [28] 或 Kirkpatrick 结构 [20]。
- $k= O(1)$ *二维近邻搜索。*数据集 S 是 R^2 中 n 个点的集合。固定整数 $k \ge 1$。给定一个点 $q \in R^{(2)}$，查询会返回 P 中最接近该点的 k 个点。q.该结构可以是建立在*阶* k Voronoi 图上的点定位结构 [20, 28]（*阶* k Voronoi 图的计算时间为 $O(n \log n)$ [6]）。在 $k = 1$ 的情况下，对于所有 $r \le n$，DDS 算法的 Time(n, r) = $O(n$ Log $r)$ [1]。不过，[1] 的算法在很大程度上依赖于在线性时间内合并两个（阶-1）Voronoi 图的能力，因此无法轻松扩展到更高的 k 值。
- *度量空间中的近似近邻搜索。*数据集 S 由一个具有恒定倍维度的度量空间（包括任何具有恒定维度的欧几里得空间）中的 n *个*对象组成。设 $dist(o_1, o_2)$ 表示空间中两个对象 o_1 和 o_2 之间的距离。给定空间中的任意对象 q，查询

返回一个对象 $o \in S$，对于所有 $o' \in S$，这样 $dist(o, q) \leq (1+\epsilon) \cdot dist(o', q)$，其中 ϵ 是一个常数。符合我们目的的结构可参见 [14, 22]。

对于 R^d 中的正交范围计数，其中 $d \geq 3$ 是一个固定常数（如第 1.2 ），节中所定义我们可以应用定理 1 得到一个有点不同寻常的结果。[3]在 $O(n \log n)$ 时间内建立一个结构，在 $Q(n) = O(n^{1-\epsilon})$ 时间内回答查询，其中常数 $\epsilon > 0$ 可以任意变小。因此，对于所有 $r \geq n^{(1)-(\epsilon)}$，定理 1 产生的 DDS 算法的 $\text{TIME}(n, r) = O(n \text{ Log } r)$。由于 ϵ 可以任意接近 0，log(r)-STREAK 界值 $\text{LOGSTREAK}(n) = n^{1-\epsilon}$ 只比最大值 n 低一个 n 的亚对数因子。如果能在所有常量维度上缩小这一差距，则意义非凡。如果能发现一种 $\text{LOGSTREAK}(n) = n$ 的 DDS 算法，那么该算法也能在 $O(n \log n)$ 时间内解决下面的*离线*版本问题：我们给定 R^d 中 n 个点的集合 P 和 n 个 *d-rectangles* 的集合 Q；目标是为每个 d-rectangles $q \in Q$ 报告 q 中覆盖了 P 中的多少个点。这个离线问题已被广泛研究，但据我们所知，最快的算法仍然需要 $n \log^{\Theta((d))()} n$ 时间。

4.2 不可分解但频谱可递变的问题

本小节将利用定理 1 来处理不可分解问题（至少不明显）。关键是要证明，对于合适的 $Q(n)$，问题是（Log n, $Q(n)$）谱可索引的。这本身就是一个有趣的课题，接下来我们将通过开发新的 DDS 算法来证明这一点，对于所有 $r \geq n$，这些算法在半平面包含、凸壳包含、范围中值和二维线性规划方面的 $\text{TIME}(n, r) = O(n \text{ Log } r)$。针对这些问题的原始算法 [11, 19] 都是采用第 2 。节回顾的自顶向下方法设计的我们的算法提供了一个对比，说明定理 1 如何促进 DDS 算法的设计。

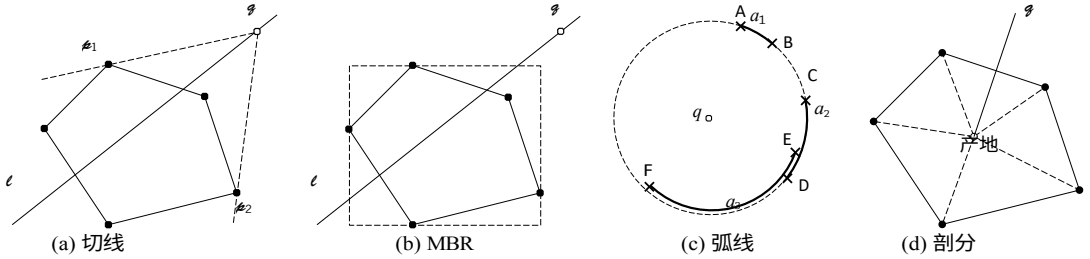
半平面包含。利用几何对偶性 [7] 可以将问题 [19] 转换为以下等价形式：

通过凸壳的直线。 S 是 R^2 中 n 个点的集合。给定 R^2 中的任意一条直线 l，通过查询可以确定 l 是否与 S 的凸壳相交，表示为 CH(S)。

我们将集中讨论上述问题。

暂且假设我们在 l 上得到一个点 q，它位于 CH(S) 外部。从 q 出发，我们可以射出两条*切线*，每条*切线*都与 CH(S) 相触，但不会进入 CH(S) 的内部。在图 1(a) 中，S 是黑色点的集合，第一条射线经过点 $p_1 \in S$，而第二条射线经过点 $p_2 \in S$。这两条射线形成一个 "楔形"，将 CH(S) 包围在其中（注意楔形的角度小于 180° ）；我们称之为 *CH(S) 上的 q 楔形*。当且仅当直线 l 穿过楔形时，直线 l 才穿过 CH(S)。如果 CH(S) 的顶点是按顺时针顺序存储的，则两条切线可以在 $O(\log n)$ 时间内找到 [25]。

我们将证明 "直线穿过凸壳 "问题是（Log n, Log n）谱可索引的。取任意整数 $s \in [1, n]$ 并设置 $m = n/s$ 。任意将 S 分成 S_1、S_2、…，$S_{(m)}$，使得 $S_i = \{s_j \text{ 为 } i \in [m]\}$，$S_{(m)} = \{s_j \mid (m-1) \in [n]\}$。要构建一个结构，需要用 $O(s \text{ Log } s)$ 的时间为每个 $i \in [m]$ 计算 CH(S_i)，并按顺时针顺序存储其顶点。结构的构建时间为 $O(n \text{ Log } s)$。让我们来看看如何用直线 l 回答查询。再次假设给出了 CH(S) 以外 l 上的一个点 q。对于每个 $i \in [m]$，在 $O(\text{Log } s)$ 时间内计算 q 在 CH(S_i) 上的楔形。从这 m 个楔形中，可以用 $O(m)$ 的时间简单地得到 q 在 CH(S) 上的楔形（我们将在讨论 "凸壳包含 "时处理一个更一般的问题）。现在，可以轻松确定 l 是否与 CH(S) 相交。到目前为止，查询时间为 $O(m \text{ Log } s)$。



(a) 切线 (b) MBR (c) 弧线 (d) 剖分

图 1 第 4.2 节中关键概念的说明

如果我们已经有了 S 的最小轴平行边界矩形，即 MBR(S)，那么就可以在 $O(1)$ 时间内找到 q。请注意，MBR(S) 必须包含 CH(S)；见图 1(b)。很明显，MBR(S_i)可以在 $O(m)$ 时间内从 MBR($S_{1)}$、MBR($S_{2)}$、...、MBR($S_{(}m$))得到而每个 MBR(S_{ij}) (i [m])可以在构建结构 时用 $O(s)$ 时间计算。因此，我们得出结论：该问题是（Log n, Log n）谱可索引的，现在可以应用定理 1。

凸壳包含。在这个问题中，S 是 R^2 中 n 个点的集合。给定任意点 q R(2)，查询就能确定 q 是否被 CH(S) 覆盖。我们将证明这个问题是（Log n, Log n）谱可索引的。

取任意整数 s [1, n]并设 m= n/s 。任意将 S 分成 S_1、S_2、...、$S_{(m)}$，使得 S_i= s for i [m 1] 和 $S(m_)$= n s(m 1)。要建立一个结构，计算每个 i [m] 的 CH($S_{(i)}$))，并按顺时针顺序存储其顶点；如前所述，这需要 $O(n$ Log $s)$ 时间。让我们看看如何回答给定点 q 的查询。对于每个 i [m]，CH(S_{ij})是否覆盖 q 可以用 O(Log s) 的时间来检查 [25]。如果任何 i 的答案为 "是"，则 q 点一定被 CH(S) 覆盖，我们就完成了。接下来的讨论假设所有 i 的 q 都在 CH(S_{ij}) 以外。如上一问题所述，在 $O(m$ Log $s)$ 时间内，计算所有 i [m] 的 q CH(S_{ij}) 上的楔形。

剩下的就是根据 m 个楔形确定 q 是否被 CH(S) 覆盖。这可以重塑为以下问题。以 q 为圆心放置一个任意圆。对于每个楔形，它的两条边界光线与圆相交成一条小于 180° 的弧。设 a_1、a_2、...、a_m 为产生的弧，并定义* 为圆上覆盖所有弧的最小弧。图 1(c) 显示了一个 m= 3 的例子。弧 a_1 被点 A 和 B 包住，弧 a_2 被点 C 和 D 包住，弧 a_3 被点 E 和 F 包住。这里，最小的弧 a^* 从点 A 顺时针延伸到 F。最重要的是，只有当*跨度至少为 180° 时，q 才会被 CH(S)1(c) ）覆盖（图 中就是这种情况--注意，如果 q 位于 CH(S) 以外，那么* 必须是由 q 在 CH(S) 上的楔形斜切而成，它必须小于 180° 。因此，我们的目标是确定* 是否至少为 180 。°

我们可以在 $O(m)$ 的时间内达到目的（即使 m 个弧的顺序是任意的）。，我们逐一处理这些弧，保持覆盖已处理弧的最小弧 a^* ，并在明确 a^* 必须至少为 180° 时停止算法。具体来说，对于 i= 1，只需将 a^* 设置为 a_1。给定下一条弧 a_i（i 2），在恒定时间内检查一条小于 180° 的弧是否能同时覆盖 a_i 和 a^* 。如果能，则更新 a^* 为该弧；否则，停止算法。例如，在图 1(c)中，处理 a_2 后，我们维护的 a^* 从 A 顺时针延伸到 D。当处理 a_3 时，算法意识到 a^* 必须至少 180° ，因此终止。

因此我们得出结论，凸壳包含问题是（Log n，Log n）谱可索引的，现在可以应用定理 1。

范围中值在这个问题中，S 是一组存储在数组 A 中的 n 个实数。

满足 1 x y n 的任意整数对 (*x*，*y*)，查询返回 *A*[*x* : *y*] 的中值。我们将证明这个问题是（Log *n*，Log *n*）谱可索引的。固定任意整数 *s* n 和 *m*= n/s。为每个 *i* 定义 *S* =*A*[(*i* 1)*s*+ 1 : *i* m 1 和 S_m=*A*[(*m*− 1)*s*+ 1 : *n*]。接下来，我们假设 *s* $\sqrt{n}$；否则，只需使用 *O*(*n* log *n*)= *O*(*n* log *s* 时间在整个 *S* 上创建一个 [11] 的结构，它能在 *O*(log *n*)= *O*(log *s*) 时间内回答 *S* 上的任何查询。

要建立一个结构 ，对每个 *i* [*m*]，按升序存储 S_i，但每个元素 S_i 都应与其在 *A* 中的的原始位置索引相关联。 可以在 *O*(*n* Log *s*) 时间内建立。让我们看看如何回答带谓词 (*x*, *y*) 的查询。首先，确定 [*m*] 中的值 *a*、*b*，使得 *A*[*x*] S_a，*A*[*y*] S_b，这可以通过以下方法实现在 *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，检查其在 *A* 中的原始索引是否位于 [*x*, *y*]）。由于 S_a 已排序，我们可以得出 S'_a 的排序时间为 *O*(*s*)。以同样的方式计算 $S'_b = S_b \cap A[x : y]$ in *O*(*s*) 时间。此时，*A*[*x* : *y*]的所有元素已被划分为 *b*− *a*+ 1 个排序数组：S'_a , $S_{(a)(+1)}$, $S_{(a)(+2)}$, ..., $S_{b\ 1}$, S'_b。现在的目标是在这些数组的结合处找到 (*y x* + 1)/2 -th 最小的元素。Frederickson 和 Johnson [10] 描述了一种从排序数组的联合中选择给定秩元素的算法。他们的算法在 *O*(*m* Log $\frac{n}{m}$) = *O*(*m* Log *s*) 的时间。总的来说，查询时间为 *O*(*s*+ *m* Log *s*) = *O*(*m* Log *s*)，因为 *s* $\sqrt{n}$。

现在我们得出结论，范围中值问题是（Log *n*，Log *n*）谱可索引的并准备应用定理 1。

二维线性规划。利用几何对偶性 [7]，可以将问题 [19] 转化为 "线穿过凸壳"（我们已经解决了这个问题）和下面的问题：

- *光线退出凸壳。*这里，*S* 是 R^2 中 *n* 个点的集合，使得 CH(*S*) 覆盖原点。给定任何一条从原点发出的射线 *q*，查询会返回 *q* 从 CH(*S*) 的边 e_{exit} 退出 CH(*S*)。

我们将集中讨论上述问题。在继续讨论之前，让我们先说明有关这个问题的两个事实：

- 给定任何射线 *q*，都可以用 [21]；中的算法在 *O*(*n*) 时间内求得 $e_{退出}$ 我们称之为*基本算法*。
- 我们可以在 *O*(*n* log *n*) 的时间内创建一个结构，从而在 *O*(log *n*) 的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线段对其进行 "剖分"；见图 1(d)。然后，就可以通过查找剖面中 q 所的三角形来回答带有射线 *q* 的查询。我们称之为*基本结构*。

与迄今为止讨论过的所有问题不同，目前我们无法证明 "射线出凸壳 "是（Log *n*，Log *n*）谱可索引的。然而，根据定理 1，我们*不必*这样做！只需证明对于任何正常数 *c*< 1，该问题是（Log *n*, n^c）可谱索引的即可。定理 1 允许我们在 *O*(*n* Log *r*) 时间内回答 *r* $^{1-c}$log n 查询。当 *r* 达到 n^{1-c} 时，我们就可以在 *O*(*n* log *n*) = *O*(*n* log *r*) 的时间内建立基本结构，从而在 *O*(log *n*) = *O*(log *r*) 的时间内回答以后的每一个查询。

这样，对于所有 *r*≤ *n*，我们都能实现 TIME(*n*, *r*)= *O*(*n* Log *r*)。

我们将证明 "射线出凸壳 "问题是（Log *n*、$\sqrt{n}$）频谱可索引。

卡普、莫特瓦尼和拉加万使用自顶向下的方法建立了具有以下性质的二叉树 。 如果节点 *u* 位于 的 层，那么 *u* 与 *S* 中 $n/2^{\ell}$ 个点的集合 *S*(*u*) 相关联。

- 树的前 层可在 *O*(*n* -) 时间内建立。

一个查询最多需要遍历一条根到叶的路径。 如果搜索过程的后裔指向一个节点 u，那么在 $S(u)$ 上运行基本算法 [21] 可以在 $O(S(u))$ 时间内找到目标边 e_{exit}。

回到我们的场景，固定任意整数 s n.在 $O(n \operatorname{Log} s) = O(n \operatorname{Log} s)$ 时间内建立 S 的前 $1+\operatorname{Log} s$ 层。要回答查询，我们沿着 $\sqrt{s}$ 的路径下降到 $\operatorname{Log} s$ 层的节点 u，如果边 e 还没有被找到。集合 $S(u)$ 最多有 $n/2^{\operatorname{Log} s} = n/\sqrt{s}$ 个点。因此，我们可以在 $S(u)$ 上运行基本算法，在 $O(n/s) = O(\sqrt{s})$ 时间内找到 e_{exit}。因此，"射线退出凸壳 "问题为 $(\operatorname{Log} n, n)$ 光谱可转位。

在本节的最后，我们要指出的是，正如上述讨论所揭示的，自上而下的方法与我们的还原之间存在着内在联系。从本质上讲，我们以增量方式构建了 [19] 的结构：*第 i 次*（i 1）"历时"（在第 3.2 ）节的证明中重新构建了 [19] [19] [19] 的前 2^{2^i} 层。 这与 不同，在 中，节点是以第 2 。节中描述的方式 "一触即扩 "的事实上，所有基于自顶向下方法设计的现有 DDS 算法都可以通过 "频谱可索引性 "的桥梁封装到我们的还原框架中，就像我们对 "射线退出凸壳 "所演示的那样。

5 使用构建时间为 $\omega(n \log n)$ 的结构的 DDS

到目前为止，我们的讨论主要集中在 $B(n) = O(\log n)$ 的数据结构上。在本节中，我们将首先说明黑盒还原必须这一条件，才能保证即使是非常数 $\log(r)$-streak 约束。第二步，我们将介绍定理 1 的扩展，即允许部署 $\max\{B(n), Q(n)\} = \operatorname{polylog} n$ *的*结构，以产生对所有 $r \le n$ 都具有良好 $\textsc{Time}(n, r)$ 的 DDS 算法。

$B(n) = \omega(\log n)$ 的恒定链边界。我们接下来的难度论证要求 n $B(n)$ 是一个凸函数。考虑任何黑箱还原算法 .回想一下 需要适用于任何具有受限数据结构的可分解问题（读者可以在继续阅读第 3.3 节之前回顾一下）。假设该算法可以保证所有此类问题都有 $\textsc{LogStreak}(n)$ 的 $\textsc{Log}(r)$-streak 约束，即 总能在 $O(n \operatorname{Log} r)$ 时间内回答 r 个查询，对于所有 $r \le \textsc{LogStreak}(n)$我们将证明，如果 $B(n) = \omega(\log n)$，那么 $\textsc{LogStreak}(n)$ 必须是 $O(1)$。

与第 3.3 ，节类似我们将提出一个可分解的问题和一个相应的数据结构。数据集 S 包含 n 个任意元素；给定任意谓词，对 S 的查询总是返回 S。每当被要求在 S 上建立一个数据结构 (S) 时，我们会故意浪费 S 的时间，然后在数组中输出 S 的任意排列。每当被要求回答查询时，我们会立即在恒定时间内返回 S。换句话说，函数 $Q(n)$ 被固定为 1。

从现在起，我们将把 r 固定为 $\textsc{LogStreak}(n)$ 的值，当给定我们设计的数据结构时，它可以确保 $\textsc{LogStreak}(n)$ 的值。我们将假设 $r = \omega(1)$；否则，$\textsc{LogStreak}(n) = O(1)$，我们的工作就完成了。由于它能在 $O(n \operatorname{Log} r)$ 时间内回答 r 个查询，因此至少有一个查询的代价是 $O(\frac{n \operatorname{Log}(r)}{r})$。我们将在下文中重点讨论这个特定查询。

的论点。

回顾第 3.3 节，要回答这个查询，算法 A 需要搜索若干（包括 0）结构 $(S_1'), (S_{(2)}'), \ldots, (S_{(t)}')$ 并扫描子集 $S_s' \subseteq S$。因为 A 需要扫描 支付 的成本因此必须认为 对于某个常量 $\alpha > 0$。我们将考虑 r（我们知道它是 $\omega(1)$）大到足以使

$\alpha - \frac{n(\text{Log})r}{r} \le n/2$。因为 A 必须服从 (5)所以，我们可以断言

$$|S_1' \cup S_2' \cup \ldots \cup S_{(t)(}'\,) | \ge |S| - |S_{(s)(}'\,)_{can}| \ge n/2。 \tag{6}$$

这意味着 $t \ge 1$。由于算法 A 必须花费恒定的时间来搜索每个结构 $\mathscr{Y}(S_i')$ $(i \in [t])$，我们必须有

$$t = \mathrm{O}((n/r) - \mathrm{Log}\, r)。 \tag{7}$$

然而，根据我们设计$\mathscr{Y}$方式，构建$\mathscr{Y}(S_1'), \mathscr{Y}(S_{(2)(}'\,)), \ldots, \mathscr{Y}(S_{(t)(}'\,))$的总成本为

$$\sum_{i \in [t]} |S_i^{(\prime)}| - B(|S_{(i)}'|). \tag{8}$$

设 $\lambda = \min_{(i)\,([]t]} |S_{(i)}^{(\prime\prime)}|$；由(6)可知 $\lambda \ge n/2$。由于 $n - B(n)$ 是一个凸函数，且 $B(n)$ 是非递减函数，我们知道当 $|S_{(i)}'| = \lambda/t$ 对于所有 $i \in [t]$ 时，(8) 最小。因此

$$(8) \ge \lambda - B(\lambda/t) \ge \frac{n}{2} - B\left(\frac{n}{2t}\right). \tag{9}$$

由 (7) 可知，$n/(2t) = \Omega(r/\log)$。因为 $B(n) = \omega(\log n)$，所以基本的渐近分析表明，$B(n/(2t))$ 必须是 $\omega(\log r)$。

现在我们得出结论：(9) 以及 (8) 必须是 $\omega(n \,\mathrm{Log}\, r)$。但这与算法 A 可以在 $O(n \,\mathrm{Log}\, r)$ 时间内处理 r 个查询的说法相矛盾。因此，我们关于 $r = \omega(1)$的假设肯定是错误的。

$B(n)$ 的 DDS 算法$= \omega(\log n)$。只要我们不执着于在 $O(n \,\mathrm{Log}\, r)$ 时间内回答 r 个查询，那么具有 $\omega(n \log n)$ 构造时间的数据结构对 DDS 仍然有用。为了使这一点正式化，我们修改了定理 1 背后的技术，以获得另一种具有以下保证的通用还原。

► 定理 **3.** *假设 $B(n)$ 和 $Q(n)$ 都是非递减函数，它们都是 $O(\log^{\gamma} n$，其中 $\gamma \ge 1$ 是一个常数。对于所有 $r \le n$，每个 $(B(n), Q(n))$ 频谱可索引问题都有一个 DDS 算法，其* $\mathrm{TIME}(n, r) = \mathrm{O}(n \,\mathrm{Log}^{\gamma} r)$。

证明类似于第 3.2 节，现移至附录 A。上述内容的一个有趣应用是 R^d中的正交范围计数/报告，其中 d 是一个至少为 3 的固定常数（参见第 1.2 ）。节中的问题定义用分数级联[9]增强的范围树在 n 个点上可在 $O(n \log^{(d)-1} n)$ 时间内构建，回答计数/报告查询也只需 $O(n \log^{(d)-1} n)$时间。该问题是可分解的，因此 $(\mathrm{Log}^{(d)-1} n, \mathrm{Log}^{(d)-1} n)$ 谱可索引。定理 3 直接给出了一种 DDS 算法，对于所有 $r \le n$，其 $\mathrm{TIME}(n, r) = O(n \,\mathrm{Log}^{(d)-1} r)$，这严格改进了第 1.2 节中提到的 [19] 结果。

参考资料

1. Alok Aggarwal 和 Prabhakar Raghavan.近邻计算的延迟数据结构问题。*信息处理通讯》（IPL）*，40（3）：119-122，1991 年。
2. 热雷米-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。接近最优内外部内存中在线多选。*离散算法》*，36:3-17，2016。
3. Jon Louis Bentley 和 Hermann A. Maurer.范围的高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 Gerth Stolting Brodal、Beat Gfeller、Allan Gronlund Jorgensen 和 Peter Sanders。迈向最佳范围中值。*理论计算机科学*》，412（24）：2588-2601，2011。

6 Timothy M. Chan and Konstantinos Tsakalidis.二维和三维计算最优确定性算法 3-D浅层切削。*离散与计算几何*》，56（4）：866-881，2016。

7 Bernard Chazelle, Leonidas J. Guibas, and D. T. Lee.几何对偶的力量。*BIT Numerical Mathematics*, 25(1):76-90, 1985.

8 Yu-Tai Ching, Kurt Mehlhorn, and Michiel H. M. Smid.动态延迟数据结构。*信息处理通讯》（IPL*），35（1）：37-40，1990 年。

9 Mark de Berg、Otfried Cheong、Marc van Kreveld 和 Mark Overmars。 *Computational 几何学：算法与应用*》。Springer-Verlag, 2008 年第 3 版。

10 Greg N. Frederickson 和 Donald B. Johnson.遴选和排序的复杂性 x+y 和有排序列的矩阵。*计算机与系统科学杂志》（JCSS*）、24(2):197-208, 1982.

11 Beat Gfeller 和 Peter Sanders.迈向最佳范围中值*国际自动机、语言和编程学术讨论会（ICALP*），第 475-486 页，2009 年。

12 Goetz Graefe, Felix Halim, Stratos Idreos, Harumi A. Kuno, and Stefan Manegold.并发自适应索引控制。*VLDB 基金会论文集》（PVLDB*），5（7）：656-667、2012.

13 Felix Halim, Stratos Idreos, Panagiotis Karras, and Roland H. C. Yap.随机数据库破解：在主内存列存储中实现稳健的自适应索引。*会议论文集 VLDB Endowment (PVLDB)*, 5(6):502-513, 2012.

14 Sariel Har-Peled 和 Nirman Kumar.低维近邻近似搜索查询。*SIAM 计算期刊*》，42(1):138-159, 2013。

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

16 Stratos Idreos, Martin L. Kersten, and Stefan Manegold.数据库破解》，第 68-78 页、2007.

17 Stratos Idreos, Martin L. Kersten, and Stefan Manegold.自组织元组重构在列存储中。*ACM 数据管理论文集》（SIGMOD*），第 297-308 页、2009.

18 Stratos Idreos、Stefan Manegold、Harumi A. Kuno 和 Goetz Graefe。合并破解之物、破解合并的内容主内存列存储中的自适应索引。*大会论文集 VLDB Endowment (PVLDB)*, 4(9):585-597, 2011.

19 Richard M. Karp、Rajeev Motwani 和 Prabhakar Raghavan.延迟数据结构。*SIAM 计算期刊*》，17（5）：883-902，1988 年。

20 David G. Kirkpatrick.平面细分中的最优搜索*SIAM 计算学报*、12(1):28-35, 1983.

21 David G. Kirkpatrick 和 Raimund Seidel.终极平面凸壳算法？*SIAM 计算杂志*》，15（1）：287-299，1986 年。

22 Robert Krauthgamer and James R. Lee.导航网：邻近性的简单算法搜索。*ACM-SIAM 离散算法年度研讨会（SODA）论文集*》、第 798-807 页，2004 年。

23 Konstantinos Lampropoulos、Fatemeh Zardbani、Nikos Mamoulis 和 Panagiotis Karras。高维度量空间中的自适应索引。*VLDB 捐赠论文集 (PVLDB*），16（10）：2525-2537，2023。

24 Rajeev Motwani 和 Prabhakar Raghavan.延迟数据结构：查询驱动的预几何搜索问题的处理。*计算研讨会论文集 几何》（SoCG*），第 303-312 页，1986 年。

25 Mark H. Overmars 和 Jan van Leeuwen.平面配置的维护 *学报 计算机与系统科学》（JCSS*），23（2）：166-204，1981 年。

26 Bryce Sandlund 和 Sebastian Wild.懒惰搜索树 *电气和电子工程师学会年会论文集 计算机科学基础（FOCS）研讨会*，第 704-715 页，2020 年。

27 Bryce Sandlund 和 Lingyi Zhang.可选择堆和最优懒搜索树。*ACM-SIAM 离散算法年度研讨会论文集》（SODA）*，第 1962-1975 页，2022 年。

28 尼尔-萨尔纳克和罗伯特-恩德雷-塔尔扬使用持久搜索树进行平面点定位 *ACM 通信》（CACM）*，29（7）：669-679，1986 年。

29 Felix Martin Schuhknecht、Alekh Jindal 和 Jens Dittrich。数据库破解的实验评估和 分析。*VLDB 期刊》*，25（1）：27-52，2016 年。

30 陶宇飞和 Dimitris Papadias。空间数据库中的范围聚合处理。*IEEE 知识与数据工程交互（TKDE）*，16（12）：1555-1570，2004。

31 Fatemeh Zardbani、Peyman Afshani 和 Panagiotis Karras.重温数据库破解的理论与实践。*扩展数据库技术（EDBT）论文集*，第 415-418 页，2020。

32 法特梅-扎尔巴尼、尼科斯-马穆利斯、斯特拉托斯-伊德雷奥斯、帕纳吉奥蒂斯-卡拉斯具有空间范围的对象自适应索引。*VLDB 基金会论文集》（PVLDB）*，16（9）：2248- 2260，2023。

33 Donghui Zhang, Vassilis J. Tsotras, and Dimitrios Gunopulos.对有范围的对象进行高效聚合。*ACM 数据库系统原理（PODS）研讨会论文集》*，第 121-132 页，2002 年。

定理 3 的证明

我们的算法以*历时*为单位执行。在*第 i (i* ≥ 1）个历元开始时，我们设置 $s = 2^{2^i}$，并创建一个结构 - B(n)，Q(n)）谱可索引性所承诺的结构

- 在 $O(n\ B(s)) = O(n\ 2^{i-\nu})$ 时间内回答 *S* 上的任何查询。该结构允许我们在以下时间内回答任何查询 $O(\frac{n}{s}$ -Q(s)) 时间。在*第* i *个*纪元*期间*回答了 *s 次*查询后，该纪元结束。这些查询的总费用为

$$s - O(\frac{n}{s} - Q(s) = O(n - Q(s)) = O(n - 2^{i-\nu}). \quad (2)$$

因此，*第 i 个*历元的总计算时间为 $O(n - 2^{i-\nu})$。由于 $\gamma \geq 1$，我们知道当 *i* 增加 1 时，$n - 2^{i-\nu}$ 至少会增加一倍。因此，所有历元的总成本将被 $O(n - 2^{h^* - \nu})$渐近支配，其中 h^* 是所需的历元数。准确地说，h^* 的值是满足两个条件的最小整数 $h \geq 1$：

- **C1**：$2^{2^h} \leq n$（*s* 的值不得超过 *n*）；
- **C2**：$\sum_{i=1}^{h} 2^{2^i} \geq r$（*h* 个历时可回答的查询次数必须至少为 *r*）。

让 *H* 代表最小整数 $h \geq 1$，使得 $2^{2^h} \geq r$。这意味着

$$2^{2^H} < r \Leftrightarrow 2^{2^H} < r^2. \quad (10)$$

当$2^{2^H} \leq n$时，上面的论证保证我们的纪元数h^* 最多为H。（不等式 $2^{2^H} \leq n$ 意味着 $2^{2^{h^*}} \leq n$，符合条件 **C1**）。在这种情况下，所有每个历时的总成本为 $O(n\ 2^{h^* - \nu}) = O(n\ 2^{(H)(-)(\nu)}) = O(n \log^{\nu} r)$.

如果 $2^{2^H} > n$，则上述论证不起作用。然而，当这种情况发生时，我们从 (10) 中可以知道，$r^2 > 2^{2^H} > n$，从而导致 $r > n$。只要 *r* 到达 n--*抢断就*会发生。 [¯]

我们用 $O(n \log^{\nu} n)$ 的时间在整个 *S* 上创建一个结构 ，并用它来回答随后的每个查询，用时为 $O(Q(n)) = O(\log^{\nu} n)$，直到 *r*= n。抢断点之后的查询需要的总费用为 $O(n \log^{\nu} n) = O(n \log^{\nu} r)$。这样，我们就得到了一种算法，在所有 *r* N 时都能保证 $\text{TIME}(n, r) = O(n \log^{\nu} r)$，从而完成了定理 3 的。证明