arXiv:2303.06288v1 [cs.DS] 11 Mar 2023

面向加权输入推广 Greenwald-Khanna 流式分位数摘要

Sepehr Assadi* Nirmit Joshi† Milind Prabhu‡ Vihan Shah §

摘要

估算分位数（如中位数或百分位数）是数据挖掘和数据科学中的一项基础任务。（流式）分位数摘要是一种数据结构，能够以流式方式处理一组 S n 个元素，并在处理结束时，对于任意 $\phi \in (0,1],$，返回一个 ϕ分位数，使其与 S 相差至多 ε 误差；也就是说，返回一个满足 $\phi' = \phi \pm \varepsilon$的 ϕ'分位数。我们尤其关注基于比较的摘要：它们只依据全序比较全集中的元素，除此之外完全不依赖全集的其他信息。目前已知最佳的确定性分位数摘要是已有 20 年历史的 Greenwald-Khanna（GK）摘要，其所需空间为 $O(1/\varepsilon)\log(\varepsilon n)$ [SIGMOD'01]。Cormode 和 Vesle´y 最近证明，这一界限对于所有确定性比较摘要而言都是最优的 [PODS'20]。

本文研究加权分位数问题，这是分位数问题的一种推广：每个元素到达时都带有一个正整数权重，表示该元素被插入的副本数。已知唯一一种通过 GK 摘要处理加权输入的方法是采用朴素做法：将每个加权元素拆分为多个未加权项，并逐个将其输入摘要中。这会导致更新时间过长，无法实际应用（与输入元素的最大权重成正比）。

我们首次对适用于加权输入的 GK 摘要进行了实质性扩展，并证明，在处理长度为 n 的数据流时（在权重取值范围及 ε 满足一些相当宽松的假设下），其所需空间为 $O((1/\varepsilon)\log(\varepsilon n))$，每个元素的更新时间为 $O(\log(1/\varepsilon) + \log\log(\varepsilon n))$。在此过程中，我们还简化了用于未加权分位数的原始 GK 摘要。

*(sepehr.assadi@rutgers.edu) 罗格斯大学计算机科学系。研究部分由美国国家科学基金会 CAREER 项目 CCF-2047061 和 Google Research 的捐赠资助。

†(nirmit@u.northwestern.edu) 西北大学计算机科学系。

‡(milindpr@umich.edu) 密歇根大学计算机科学与工程系，安娜堡。§(vihan.shah98@rutgers.edu) 罗格斯大学计算机科学系。研究部分由美国国家科学基金会 CAREER 项目 CCF-2047061 资助。部分工作完成于作者在罗格斯大学卡姆登分校就读本科期间，并部分获得美国国家科学基金会项目 CCF-1910565 资助。

目录

1 引言 1
1.1 我们的贡献.2 1.2 进一步的相关工作.2

2 基本设置（非加权设置）nweighted Setting) 3
2.1 分位数摘要.3 2.2 时间步和带.4 2.3 对 (r-最小值，r-最大值) 的间接处理及主要不变量.6

3 一种贪心 $O(\frac{1}{\varepsilon}\cdot$ $\log^2(\varepsilon n))$ 大小摘要 7
3.1 空间分析.8 3.2 算法 1 的高效实现.10 4 最终 $O(\frac{1}{\varepsilon}\cdot$

$\log(\varepsilon n))$ 大小摘要 12
4.1 空间分析.13 4.2 算法 2 的高效实现.16

5 基本设置（加权设置）
17 5.1 时间步和带.19 5.2 间接处理 r-min 和 r-max 值.19 6 一种非平凡的扩展

16.1 空间分析.22 6.2 算法 3 的高效实现.25 适用于加权数据流的 GK 算法扩展 2

一种贪心算法 用于加权数据流的算法 2 9
A.1 空间分析.30 A.2 算法 4 的高效实现.32

1 引言

给定全序宇宙中的一组元素 S x_1、……、 x_n，该宇宙中元素 x 的秩记作 rank(x)，表示 S 中满足 $x_j \leqslant x$ 的元素 x_j 的数量。类似地，对于任意 $\phi \in$ （$0, 1]$，， S 的 ϕ 分位数是秩为 $= \lceil \phi \cdot n \rceil$ 的元素 $x_i \in S$（rank(x_i)）。计算分位数是一个基础问题，应用范围广泛，因为分位数能够简洁地表示输入元素的分布。本文始终仅关注该问题的基于比较的算法：这类算法只能根据元素的顺序比较全序宇宙中的两个元素，除此之外对该宇宙一无所知。

我们关注由 Alon、Matias 和 Szegedy 的开创性工作 AMS [提出的96] 数据流 S 模型中的分位数估计问题。在该模型中， S 的元素以任意顺序逐个到达，流算法只能对这些数据进行一次扫描，并且只能使用有限内存，因此无法直接存储 S 的全部内容。早在四十多年前，Munro 和 Paterson 就证明了在数据流模型中无法精确地解决这个问题 [MP78] ，因此研究重点转向寻找近似算法：给定 $\varepsilon > 0$，算法可以返回一个 ε 近似 ϕ分位数，即一个（$\phi \pm \varepsilon$）分位数。更正式地说，我们关注以下数据结构：

定义 1.1（分位数摘要）。一个 ε-近似分位数摘要以数据流方式处理任意一组元素，并在处理结束时，针对任意给定的分位数 $\phi \in$ （$0, 1]$,定义为任意 ϕ'-分位数，其中 $\phi' \in [\phi - \varepsilon, \phi + \varepsilon]$），求出一个 ε-近似的 ϕ-分位数。

如果不考虑问题中的流式特性，总能在 $O(1/\varepsilon)$ 空间内计算出一个 ε-近似分位数摘要；只需存储来自 S 的 ε 分位数、 3ε 分位数、 5ε 分位数，依此类推。显然，对于任意给定的 ϕ，返回与其最接近的已存储分位数，就能得到一个 ε-近似的 ϕ 分位数。同样显而易见的是，对于该问题而言，这一空间需求在信息论意义上是最优的。然而，由于事先并不清楚如何计算 S 所需的分位数，因此无法在流模型中直接实现这一方法。

第一个（流式） ε-近似分位数摘要由 Manku、Rajagopalan 和 Lindsay [MRL98] 提出。MRL 摘要使用 $O((1/\varepsilon) \log^2(\varepsilon n))$ 空间，并要求事先知道数据流的长度。此后不久，Greenwald 和 Khanna [GK01] 对其进行了改进，提出了 GK 摘要，使用 $O((1/\varepsilon) \log(\varepsilon n))$ 空间，且不再需要知道数据流的长度。这是确定性比较摘要领域的最先进成果。通过引入随机化，还可以进一步降低这些算法的空间需求，并实现不依赖数据流长度的界。随机摘要领域的最先进成果是 Karnin、Lang 和 Liberty [KLL16] 提出的算法：它使用 $O((1/\varepsilon) \log \log (1/\varepsilon\delta))$ 空间，以至少 $(1 - \delta)$ 的概率构造一个 ε-近似分位数摘要。我们将在第 1.2 节中更详细地讨论随机摘要和非比较型摘要相关文献。

虽然随机化能使流式算法更加节省空间，但这类算法大多有一个主要缺点：其分析关键依赖于这样一个假设，即输入流独立于算法所用的随机性。在一些场景中，这一假设并不现实；例如，算法的未来输入取决于其先前的输出。近来，人们因此开始关注能够抵御对手攻击的算法，即使对手可以自适应地选择数据流，这些算法仍能正常工作 [MNS11、GHR+12、HW13、ABED+21、GHS+12、NY15]。确定性算法天然能够抵御对手攻击，因此，理解这类算法本身就是一个值得研究的目标。

本文聚焦于确定性摘要，具体而言，旨在加深我们对 GK 摘要的理解。多年来，围绕它们出现了两个重要问题：能否改进GK 摘要的空间开销，甚至将其降至信息论最优界O（$1/\varepsilon$）？ 此外，能否以一种更简洁的方式简化GK 摘要及其复杂的分析，从而更便于提出和研究这些摘要的推广形式？ （参见“次线性算法开放问题列表”中的问题 2[wik] ，由 Cormode 提出；另见 [KLL16、CV20、LWYC16、ACH+12] 对这一问题的类似变体，例如输入项带有权重时的情形）。

第一个问题最初由 Hung 和 Ting [HT10] 提出，他们证明了一个 $\Omega((1/\varepsilon)\log(1/\varepsilon))$ 的空间下界，适用于 ε-近似分位数摘要，改进了信息论下界。最近，Cormode 和 Vesle´y [CV20] 彻底解决了这个问题，他们证明 GK

摘要在渐近意义上是最优的：对于任何确定性（基于比较的）摘要，都需要 $\Omega((1/\varepsilon)\log(\varepsilon n))$ 的空间。不过，上述第二个问题仍未得到令人满意的解决。本文在回答这个问题上取得进展，证明 GK 摘要可以推广以处理加权输入。具体而言，我们提出了用于构建以下数据结构的算法。

定义 1.2（加权分位数摘要）。考虑一个加权数据流 S_w，其中包含 n 次更新（x_i，$w(x_i)$），适用于 $1 \leqslant i \leqslant n$。第 i 次更新表示插入 $w(x_i)$ 个元素 x_i 的副本（保证权重 $w(x_i)$ 为正整数）。我们将 $W_k = \sum_{i=1}^{k} w(x_i)$ 定义为 S_w 中前 k 个元素的权重总和。一个 ε-近似加权分位数摘要是一种数据结构，它对 S_w 进行单遍处理，并在结束时，对于任意 $\phi \in [0,1)$，找到一个 x_j，使得

$$\left(\sum_{x_i<x_j} w(x_i),\ w(x_j) + \sum_{x_i<x_j} w(x_i)\right] \cap \left[(\phi-\varepsilon)W_n, (\phi+\varepsilon)W_n\right] \neq \emptyset. \tag{1}$$

1.1 我们的贡献。

构造加权分位数摘要的一种方法，是将每个加权项拆分为多个未加权项，再将它们输入未加权摘要，例如 GK 摘要。然而，这类算法速度较慢，因为处理一个元素所需的时间与其权重成正比。因此，人们一直在探寻是否存在更快的算法。我们对此给出了肯定的回答，并在 第 6 节中提出了一种解决这一问题的快速算法。具体而言，当权重为 poly($O((1/\varepsilon)\log(\varepsilon n))$ 时，该算法使用 $O(\log(1/\varepsilon)+\log\log(\varepsilon n))$ 的空间，并且每处理一个元素需要 n 的更新时间；对于长度为 n 的数据流，且 $\varepsilon \geqslant \frac{1}{n^{1-\delta}}$ 对任意 $\delta \in (0,1)$ 均成立（定理 3）。这与 GK 摘要用于汇总包含 n 个未加权项的 [GK01、LWYC16] 数据流时的空间和时间复杂度相匹配。据我们所知，这是对 GK 算法在加权数据流上的首次（非平凡）扩展。

为此，我们还通过简化或完全绕过 GK 摘要中若干较复杂的组成部分，提出了对其的一种新描述，例如第 4 节中的所谓“树表示”以及复杂的“压缩”操作。作为预热，我们还在第 3 节给出了一种用于未加权分位数的简单贪心算法，并证明它需要 $O((1/\varepsilon)\log^2(\varepsilon n))$ 空间（我们还在附录 A 中将该算法扩展到加权设置）。尽管该算法的空间界并非最优，但它有助于阐释 GK 摘要并提供直观理解。有趣的是，这种摘要与所谓的 GKAdaptive 摘要 [LWYC16] 颇为相似（尽管并不完全相同）；GK 已将其作为 GK 摘要的一种更实用的变体提出（Luo 等人 [LWYC16]进一步通过实验表明，GKAdaptive 的表现优于 GK 摘要）。虽然目前尚无关于 GKAdaptive 的理论保证，但我们证明，对该算法稍作修改后，便可通过简单分析得到 $O((1/\varepsilon)\log^2(\varepsilon n))$ 空间上界（定理 1）。

我们还强调，与原始 GK 摘要类似，我们的加权扩展不需要预先知道数据流的长度。这一保证意味着，我们可以在整个数据流过程中持续跟踪分位数，其误差与数据流当前的总权重成正比，而不只是到数据流结束时才能跟踪。

1.2 更多相关工作

流模型中的分位数估计已从多个角度得到广泛研究；感兴趣的读者可参阅 [MP78、GM09]，了解多遍算法的相关结果；参阅 [GM09、CJP08]，了解随机到达数据流；参阅 [CKMS06、GZ03]，了解偏置分位数；以及参阅 [ACH+12]，了解可合并摘要，以上仅为若干代表性示例。下文将讨论范围限定在这一问题最基本的变体：对一个对抗性排序数据流进行单遍处理，获得一个 ε-近似摘要。

我们已经讨论过 [GK01、MRL98] 中的确定性摘要以及 [HT10、CV20] 中的下界（另见 [LWYC16] 对这些结果的实验研究）。现在我们来讨论随机摘要方面的文献。简单应用 Chernoff 界可知，从数据流中抽取 $O((1/\varepsilon^2)\log(1/\varepsilon))$个元素，即可在较高的常数概率下得到一个 ε-近似分位数摘要。假设我们知道数据流的长度，就可以预先抽取这些元素，并将它们输入 GK 摘要，从而得到一个空间复杂度为 $O((1/\varepsilon)\log(1/\varepsilon))$的随机化算法（见 [MRL99]）。Felber 和 Ostrovsky 在 [FO15] 中展示了如何在这一方法中去除必须预先知道数据流长度的假设

。最后，Karnin、Lang 和 Liberty 为这个问题设计了一种大小为 $O((1/\varepsilon)\log\log(1/\varepsilon))$ 的随机摘要，达到了当前最先进水平（ [KLL16] 算法能够以恒定成功概率，在 $O(1/\varepsilon)$ 空间内回答单个分位数查询，这一空间复杂度是最优的；随机摘要的更多下界结果见 [KLL16,CV20] ）。 [ILL+19]提出了该算法针对加权分位数问题的扩展。该扩展使用 $O((1/\varepsilon)\cdot\sqrt{\log(1/\varepsilon)})$ 空间，并以每个~~元素 $O(\log(1/\varepsilon))$~~ 的更新时间，高概率地输出所有 ε-近似分位数。

与这条研究路线并行，研究人员也研究了受限全集上的非基于比较的摘要（例如，参见 [CM04、SBAS04、BLRV13]）。例如，Shrivastava 等人 [SBAS04] 设计了一种简单的确定性 ε近似分位数算法，其所需空间为 $O(\frac{1}{\varepsilon}\log|U|)$，适用于已知的元素全集 U。该算法的空间界与 GK 摘要（以及我们的结果）无法比较，并且 [HT10、CV20] 的下界不再适用于这些摘要。需要指出的是，使用这些非基于比较的摘要需要预先知道全集 U，因此它们不适合用于跟踪浮点值或字符串数据流上的分位数。

2 基本设置（无权设置）

我们首先介绍无权重分位数摘要的基本设置和一些预备定义。随后，在接下来的两节中，我们将展示如何在流模型中以有限的空间和较快的更新时间维护这些摘要。需要指出的是，我们的摘要与 [GK01], 的摘要非常相似，主要区别在于数据流过程中维护摘要的算法以及相应的分析。为确保内容完整，正文中可能会重复介绍一些内容。

2.1 分位数摘要

我们的摘要记为 分位数摘要，本质上是一个集合，其中包含数据流中出现的元素，以及为每个元素存储的一些元数据。我们用 s 表示存储在 分位数摘要 中的元素数量，并用 e_i表示任意 $i\in[s]$ 的第 i小元素（按从小到大排列）在 分位数摘要 中的位置（当元素的秩无关紧要时，我们用 e 表示 分位数摘要 中的任意元素）。每个元素所存储的元数据也只占 O（1）个字，因此摘要的总大小为 O（s）个字。具体而言， e 在 分位数摘要 中，每个元素的主要信息如下（额外存储的信息将在 第 2.2 节中定义）：

- r-最小值(e) 和 r-最大值(e)：分位数摘要对 rank(e) 维护的上下界，即该元素在目前为止数据流中出现的元素里的秩（由于我们没有存储所有元素，无法确定已存储元素的确切秩，因此重点是维护恰当的上下界）。为方便记号起见，本文中均令 r-最小值(e_0) $=0$ 。

为便于说明并消除边界情况，我们在数据流开头插入一个 $+\infty$ 元素，该元素被视为大于其他任何元素，并在分位数摘要中将其存储为 e_s。该元素的 r-最小值和 r-最大值也始终等于已访问元素的数量（包括该元素自身）。由于 $+\infty$ 是最大元素，将其插入 S 不会影响其他任何元素的秩。

在数据流处理过程中，我们需要能够向摘要中插入一个元素或删除一个元素，具体操作如下（流式算法的主要任务是在数据流处理过程中决定插入哪些元素以及删除哪些元素）；参见 图1 中的示意图。

Insert(x)。将给定元素 x 插入分位数摘要。

(i) 查找分位数摘要中满足 $e_i>x$的最小元素 e_i；

(ii) 将 r-最小值(x) 设为 $=$ r-最小值(e_{i-1}) $+1$，并将 r-最大值(x) 设为 $=$ r-最大值(e_i)；此外，对所有 $j\geqslant i$，将 r-最小值(e_j) 和 r-最大值(e_j) 各增加一。将元素 x 存储在分位数摘要中。

删除(e_i)。从分位数摘要中删除元素 e_i 。

(i) 从分位数摘要中移除元素 e_i；保持所有其余的 r-最小值、r-最大值不变。

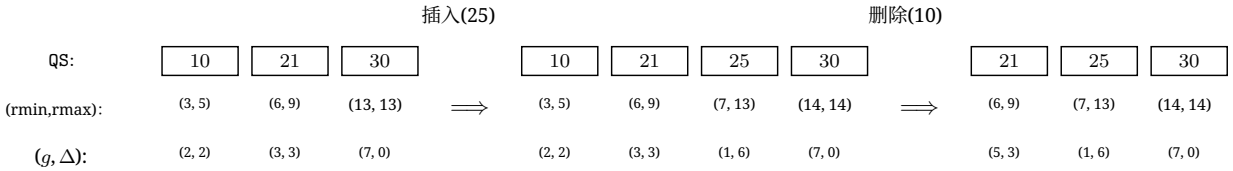


图1： 从某个任意状态开始，摘要中更新操作的示意图（本图中的参数（g， Δ）定义于第 2.3 节）。

不难验证，这两种操作都能正确维护 r-最小值和 r-最大值。下面的论断表明，只要我们能够确保 r-最小值和 r-最大值具有“适当”的界，就能正确回答分位数查询。

命题 2.1。假设分位数摘要是长度为 n 的无权流的摘要，并且对所有 $i \in [s]$都满足 r-最大值(e_i) –r-最小值(e_{i-1}) $\leqslant \varepsilon n$ ；那么，分位数摘要是一个 ε-近似分位数摘要。

证明。给定任意 ϕ，令 $r = \lceil \phi \cdot n \rceil$。我们将证明分位数摘要始终包含一个元素 e，其秩属于$[r - \varepsilon n, r + \varepsilon n]$，即 r-最小值($e$) $\geqslant r - \varepsilon n$ 且 r-最大值(e) $\leqslant r + \varepsilon n$。因此，返回该元素作为答案，就会得到一个 ε-近似的 ϕ-分位数。

对于任意 $r \leqslant \varepsilon n$，由引理的条件可知，r-最小值($e_1$) $\geqslant 0 \geqslant r - \varepsilon n$ 且 r-最大值(e_1) 6 r-最小值(e_0)$+\varepsilon n \leqslant r + \varepsilon n$，因此我们可以返回 e_1。否则，令 e_j为满足 r-最小值(e_j) $\geqslant r - \varepsilon n$的最小元素，因此 r-最小值($e_{j-1}$) $< r - \varepsilon n$；同样，由引理的条件可知，

$$\text{r-max}(e_j) \leqslant \text{r-min}(e_{j-1}) + \varepsilon n < r - \varepsilon n + \varepsilon n \leqslant r + \varepsilon n,$$

因此我们可以在此处返回 e_j。 ∎

我们后续各节中算法的主要目标是更新 分位数摘要，使得 Claim2.1 的条件得以满足，同时无需存储过多元素。

2.2 时间步和带

另一个重要概念是时间步 和 带。本节中，我们先为无权设置定义这些概念，然后在此基础上于第 5 节中为加权设置定义它们。我们以目前为止数据流中出现的元素数的 $\Theta(1/\varepsilon)$倍来衡量时间。形式化地说，

定义 2.2（时间步）。令 $\ell := \frac{1}{\varepsilon}$，并假设它是一个整数。我们将数据流划分为大小为 ℓ 的连续数据块；时间步 t 指的是第 t 个元素数据块，记为 $(x_1^{(t)}, \ldots, x_\ell^{(t)})$（我们假设数据流的长度是 ℓ 的倍数）[1]。我们将 $t_0(x)$ 定义为 x 在数据流中出现时所在的时间步。

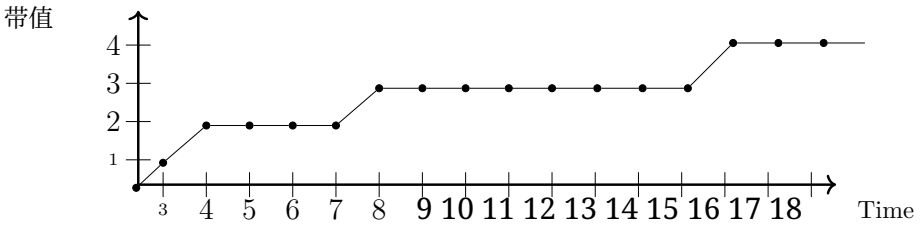
接下来要介绍的重要定义是带值和带，借鉴自 [GK01]。粗略地说，我们希望能够将数据流中的元素划分为“少量”组（带），使得同一组内元素的插入时间“几乎相同”（以此作为其 r-最小值估计精度的替代指标，以及 r-最大值估计精度的替代指标）。更多详情请参见第 2.2 节。形式化地说，

定义 2.3（带值与带）。对于无权流中的任意元素 x，我们按如下方式为其分配一个称为带值的整数，记作 b 值(x)：

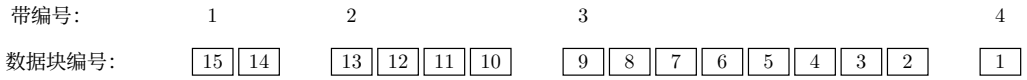
(i) *At the time step* $t = t_0(x)$, *we set* $\text{b-value}(x) = 0$;

(*ii*) 在任意时间步 $t > t_0(x)$，如果 t 是 $2^{\text{b-value}(x)}$ 的倍数，那么我们就将 b 值(x) 增加 1。

[1]此定义中的两个假设均不失一般性：我们可以将 ε 的值乘以一个 $O(1)$ 因子，以确保满足第一个假设；并在末尾添加 $O(1/\varepsilon)$ 个虚拟元素，以确保满足第二个假设。



(a) 时间步 2 插入的元素的带值的变化。



(b) $t = 15$ 的带值分布。

图2：带值与带的示意图。

定义 2.3 基本上是根据元素的值 $t - t_0(x)$ 在每个时间步进行的一种几何分组t (参见 图2)。这种分组与典型几何分组方法的唯一不同之处在于，它会随时间保持“稳定”：在某个时刻进入同一带的元素，此后会一直保持相同的带值（尽管带值可能随时间变化）。以下两个观察对此进行了形式化说明。

观察 2.4。对于元素 x，在任意 $t \geqslant t_0$（x）时，$2^{\text{b-value}(x)-1} - 2 \leqslant t - t_0$（$x$）$\leqslant 2^{\text{b-value}(x)+1}$。

这仅仅是因为，每当 b 值(x) 变为某个 α时，其带值再次增加 1 至少需要 x，至多需要 $2^{\alpha-1}$ 个 2^{α} 时间步（根据定义 2.3）。因此，

$$t - t_0(x) + 1 = \sum_{\alpha=0}^{\text{b-value}(x)} \#\text{ of time steps with band-value } \alpha \geqslant \sum_{\alpha=1}^{\text{b-value}(x)-1} 2^{\alpha-1} = 2^{\text{b-value}(x)-1} - 1;$$

$$t - t_0(x) + 1 = \sum_{\alpha=0}^{\text{b-value}(x)} \#\text{ of time steps with band-value } \alpha \leqslant \sum_{\alpha=0}^{\text{b-value}(x)} 2^{\alpha} = 2^{\text{b-value}(x)+1}.$$

观察 2.4的推论是，在时刻 t 带值的 $B^{(t)} = O(\log t$，且在任意时刻，属于带 0 至 α 的元素总数至多为 $\ell \cdot 2^{\alpha+1}$（因为对于任意元素 x，其 b 值(x) $\leqslant \alpha$，$t - t_0(x) \leqslant 2^{\alpha+1}$，因此只有最近 $2^{\alpha+1}$ 步内到达的元素可能属于带 0 至 α，且每个时间步包含 ℓ 个元素）。我们将其记录为以下方程，以供后续参考：

$$\#\text{ of band values } B^{(t)} = O(\log t) \quad \text{and} \quad |\text{Band}_{\leqslant\alpha}| \leqslant \ell \cdot 2^{\alpha+1} \text{ for all } \alpha \geqslant 0. \tag{2}$$

观察 2.5。假设在时间 t，元素 x、y 的 b 值分别为 b-value(x) 和 b-value(y)，且 b-value(x) 6 b-value(y)；那么，在任意时间步 $t' \geqslant t$，b-value(x) 6 b-value(y)。

这是因为元素的带值会根据当前时间步的值（而非相对数值）同时更新。

现在我们将证明一个引理，以便快速计算元素的 b 值。

引理 2.6。对于元素 e 和整数 $\alpha \geqslant 1$，在时间步 t 之后，b 值(e) $= \alpha$ 当且仅当满足：

$$2^{\alpha-1} + (t \bmod 2^{\alpha-1}) \leqslant t - t_0(e) < 2^{\alpha} + (t \bmod 2^{\alpha}). \tag{3}$$

证明。定义 $I_{\alpha,t} = [2^{\alpha-1} + (t \bmod 2^{\alpha-1})\ 2^{\alpha} + (t \bmod 2^{\alpha}))$. 我们首先对 t 使用归纳法证明：对于 b 值(e)为 $= \alpha$ 的元素 e， $t - t_0(e) \in I_{\alpha,t}$。当 e 首次插入摘要时，其 b 值为 0，并在紧接着的时间步被提升至带 $_{1}$。考虑时间 t，即元素 e 的插入时间之后一个时间步，可以很容易验证，$(t - t_0(e)) = 1$ 位于区间 $I_{1,t_0(e)}$。假设该结论对 e 在截至时间 t 时均成立。设元素 e 在时间 t 的 b 值为 α。

如果 $(t+1)$ 是 2^{α} 的倍数，b 值(e)会增加到 $\alpha + 1$。我们希望证明 $(t+1) - t_0(e)$ 属于区间 $I_{\alpha+1,\ t+1}$。我们注意到

$$\begin{aligned}(t+1) - t_0(e) &\in [2^{\alpha-1} + 1 + (t \bmod 2^{\alpha-1}),\ 2^{\alpha} + 1 + (t \bmod 2^{\alpha})) && \text{(Induction hypothesis)}\\ &= [2^{\alpha}, 2^{\alpha+1}) && (\ (t+1) \text{ is a multiple of } 2^{\alpha})\\ &= I_{\alpha+1,t+1}.\end{aligned}$$

如果 $(t+1)$ 不是 2^{α} 的倍数，b 值(e)不变。我们将证明在这种情况下，$(t+1) - t_0(e)$ 属于 $I_{\alpha,\ t+1}$。根据归纳假设，

$$\begin{aligned}(t+1) - t_0(e) &\leqslant 2^{\alpha} + 1 + (t \bmod 2^{\alpha})\\ &= 2^{\alpha} + ((t+1) \bmod 2^{\alpha}). && ((t+1) \text{ is not a multiple of } 2^{\alpha}.)\end{aligned}$$

为证明下界，我们注意到

$$\begin{aligned}(t+1) - t_0(e) &\geqslant 2^{\alpha-1} + 1 + (t \bmod 2^{\alpha-1})\\ &\geqslant 2^{\alpha-1} + ((t+1) \bmod 2^{\alpha-1}). && ((t \bmod 2^{\alpha-1}) + 1 \geqslant (t+1) \bmod 2^{\alpha-1}))\end{aligned}$$

这表明 $(t+1) - t_0(e)$ 属于区间 $I_{\alpha,\ t+1}$。

引理的另一个方向可由以下事实推出：对于固定的 t，区间 $I_{\alpha,\ t}$ 互不相交，并且 $(t - t_0(e))$ 总是属于区间 $I_{\text{b-value}(e),\ t}$。 ∎

容易看出，利用 引理 2.6，我们可以高效地计算元素在 O中所处的带，时间为 (1)。我们在下文记录这一观察。

观察 注 2.7。在给定 $t_0(e$ 的情况下，可以在时间 t 内计算时刻 t 的 b 值(e)，所需时间为 $O(1)$).

2.3 对 (r-最小值,r-最大值) 的间接处理与主要不变量 t

事实证明，正如[GK01], 所观察到的，与其直接处理 r-最小值，r-最大值，不如处理以下两个参数，这样会容易得多:

$$g_i := \text{r-min}(e_i) - \text{r-min}(e_{i-1}) \qquad \Delta_i := \text{r-max}(e_i) - \text{r-min}(e_i); \tag{4}$$

显然，r-最小值$(e_i) = \sum_{j=1}^{i} g_j$ 且 r-最大值$(e_i) = \Delta_i + \sum_{j=1}^{i} g_j$。因此，为我们的目的，对每个元素存储 (g, Δ) 值等价于存储其 r-最小值和 r-最大值。于是，我们要求算法满足的主要不变量如下。

不变量 1。在任意时刻 t，对于分位数摘要 QS 中的任意元素 $e_i \in$，都有 $g_i + \Delta_i \leqslant t$。

根据 公式 (4)， $g_i + \Delta_i =$ r-最大值(e_i) − r-最小值(e_{i-1})。因此，由于 $t = n/\ell = \varepsilon n$ 对于长度为 n 的数据流，任何维护不变量 1 的算法都是一个 ε-近似分位数摘要，这由命题 2.1得出。

现在简要指出使用（g， Δ）值的好处。首先，以下观察说明了这些值如何随着摘要中的操作而变化（参见图1）。

观察 2.8。在分位数摘要 QS 中:

- 插入(x): 将 $g(x) = 1$ 和 $\Delta(x) = g_i + \Delta_i - 1$ 设为相应值，并保持其余 (g, Δ) 值不变（此处，(g_i, Δ_i) 指过程定义的 e_i 的对应值）；

- 删除(e_i)：设置 $g_{i+1} = g_{i+1} + g_i$，并保持其余 (g, Δ) 值不变 .

这一观察的证明直接推论自公式 (4) 以及这些操作如何改变 r-最小值，r-最大值。现在，我们可以将每个 g, Δ 值解释如下：
g值。我们将新插入元素的 g值设为 1。此后，只有在 e_{i-1} 被删除时， e_i 在分位数摘要中的元素 g值才会改变，结果为 $g_i = g_i + g_{i-1}$。当 e_{i-1}从摘要中删除时，我们称 e_i 覆盖了 e_{i-1}；此时， e_i 也会覆盖此前由 e_{i-1} 覆盖的所有元素（每个元素在插入时只覆盖自身）。我们定义

- $C(e_i)$：由 e_i覆盖的元素集合。根据定义，在任意时刻，

$$g_i = |C(e_i)| \text{ and } C(e_i) \cap C(e_j) = \emptyset \tag{5}$$

对于当前存储在分位数摘要中的任意 e_i、 e_j。

请注意，对于任意元素 e_i， g_i 表示已从数据流中删除、现由 e_i覆盖的元素数量。因此，对于存储在分位数摘要中的某个元素，的值较大虽然会给维护不变量 1带来挑战，但这必然意味着数据流中的许多元素已从分位数摘要中删除。这一关键性质有助于我们界定分位数摘要的大小。

Δ-值。 我们将 Δ-值设为新插入元素的 $g_i + \Delta_i - 1$ （其中 e_i 是大于插入元素的 分位数摘要 中的最小元素）。此后，元素的 Δ-值不再改变（不同于g-值）。维护 不变量 1 （针对元素 e_i）后，我们便可以得出：

$$\Delta(x) \leqslant t_0(x). \tag{6}$$

因此，元素越早插入数据流，其最坏情况下的 Δ-值就越小。结合前文所述带的几何分组特性，这意味着在任意时刻 t，具有“相对较小” Δ-值的元素数量（相较于 t，即 不变量 1当前的目标值）几乎呈指数级多于具有“较大” Δ-值的元素数量。这是帮助我们以较小空间维护不变量 1 的第二个主要性质。

3 贪心 $O\left(\frac{1}{\varepsilon} \cdot \log^2(\varepsilon n)\right)$ 大小摘要

作为介绍主要算法的预热，我们首先提出一种非常简单且贪心的分位数摘要更新方法 分位数摘要 ，以在 不变量 1 下维持 $O(\frac{1}{\varepsilon} \cdot \log^2(\varepsilon n))$ 的空间复杂度。在任意一步中，算法首先将所有到达的元素插入 分位数摘要。然后，在 删除步骤中，只要存在某个元素，删除它不会违反 不变量 1 （以及关于 b 值的另一个简单条件），算法就将其从 分位数摘要中删除。

算法 1。用于更新分位数摘要的贪心算法。

对于每个包含到达元素 $(x_1^{(t)}, \ldots, x_\ell^{(t)})$ 的时间步 t：

（i）对数据块中的每个元素运行 Insert($x_j^{(t)}$)。

（ii）反复对满足以下条件的任意（任选）元素 e_i 执行 Delete(e_i)：

$$(1)\ \text{b-value}(e_i) \leqslant \text{b-value}(e_{i+1}) \qquad \underline{\text{and}} \qquad (2)\ g_i + g_{i+1} + \Delta_{i+1} \leqslant t$$

定理 1。对于任意 $\varepsilon > 0$ 和长度为 n 的数据流，算法 1 可在 $O(\frac{1}{\varepsilon} \cdot \log^2(\varepsilon n))$ 的空间内维护一个 ε-近似分位数摘要。此外，算法 1 存在一种实现，对每个元素的最坏情况处理时间为 $O\big(\log(1/\varepsilon) + \log\log(\varepsilon n)\big)$。最后，每次分位数查询的最坏情况时间为 $O\big(\log(1/\varepsilon) + \log\log(\varepsilon n)\big)$。

算法 1 维持 不变量 1 因为它可能只会删除一个元素 e_i 当 $g_i + g_{i+1} + \Delta_{i+1} \leqslant t$时，根据 观察 2.8 可知，这意味着 $g_{i+1} + \Delta_{i+1} \leqslant t$ 在 g删除后。其他（g， Δ）值保持不变。

正如 第 2.3 节中所述，维持 不变量 1 直接意味着 分位数摘要 在整个数据流中都是 ε近似分位数摘要。现在，我们来界定该算法下 分位数摘要 的大小，从而向证明 定理 1迈进一步。然后，在 第 3.2 节中，我们将展示如何快速实现该算法。

注释 3.1. GKAdaptive 算法与 [LWYC16] 和算法 1的区别在于，GKAdaptive 在尝试删除元素时只检查条件 (2)（见算法 1）。

3.1 空间分析

在本小节中，我们证明算法 1所使用的空间上界。具体而言，我们有以下结论：

引理 3.2. 对于任意 $\varepsilon > 0$ 以及长度为 n的数据流，算法 1 可在 ε空间内维护一个 $O(\frac{1}{\varepsilon} \cdot \text{对数}^2(\varepsilon n))$ 近似分位数摘要。

在时间 t执行删除步骤后，分位数摘要QS 中存在的任意元素 e_i 要么满足 b 值(e_i) >b 值(e_{i+1})，要么满足 $g_i + g_{i+1} + \Delta_{i+1} > t$；否则算法 1 就会删除该元素。我们将分位数摘要QS 中满足前一个条件的元素称为 1 型元素，将仅满足后一个条件的元素称为类型 2元素。因此，每个元素恰好属于这两种类型之一（但只有 $e_s = +\infty$的情况可以忽略）。

我们首先证明，1 型元素的数量不会比 2 型元素多出太多。这是因为连续 1 型元素的带值会从一个元素到下一个元素严格递减，因此相邻的 1 型元素不可能很多。

引理 3.3. 在时间步 t 的删除步骤之后，分位数摘要中存储的 1 型元素数量至多是 2 型元素数量的 $O(\log t)$ 倍。

证明。 设 e_i、e_{i+1}、.....、e_j是分位数摘要中一串 连续的元素，它们全都是 1 型。根据 1 型元素的定义，b 值(e_i) >b 值(e_{i+1}) > . . . >b 值(e_j)。由于时间 t 时的带值数量为 $B^{(t)} = O(\log t)$，根据 公式 (2)，这串序列的长度至多为 $O(\log t)$。因此，每 $O(\log t)$ 个 1 型元素中，至少有一个 2 型元素，这便直接推出了该引理。 ∎

引理 3.3 使我们在下文中只需对 2 型元素的数量给出上界，这也是本节证明的主要部分。

引理 3.4。 在时间步 t 的删除步骤之后，分位数摘要中的类型 2 元素数量为 $O(\ell \cdot \log t)$.

证明。 证明的第一步是任取一个 2 型元素 e_i 并对 $g_i + g_{i+1}$给出下界；直观而言，这个数应该“很大”，因为我们没有删除 e_i 的唯一原因是删除步骤的条件 (2)，见 算法1。

命题 3.5。在时间步 t 的删除步骤之后，对于任意 2 型元素 e_i, $g_i + g_{i+1} \geqslant 2^{\text{b-value}(e_{i+1})-1} - 2$。

证明。 由于 e_i 是类型 2， e_i 未从 分位数摘要 中删除的唯一原因是 $g_i + g_{i+1} + \Delta_{i+1} > t$。 此外，回顾一下，$\Delta_{i+1} \leqslant t_0(e_{i+1})$ 根据 公式 (6)，因此，

$$g_i + g_{i+1} > t - t_0(e_{i+1}) \geqslant 2^{\text{b-value}(e_{i+1})-1} - 2,$$

其中第二个不等式依据观察 2.4，该观察指出，元素需要指数时间才能到达 α 带α。 Claim3.5 ∎

命题 3.5 给出了每个 2 型元素 g的 b 值上界 e_i，其上界是 g的 b 值和带值的函数下一个 元素 e_{i+1}。基于此，我们将类型 2 元素划分为 $B^{(t)}$ （可能带的数量）个集合 X_0, . . ., $X_{B^{(t)}}$ 其中对于任意带值 α:

$$X_\alpha := \{e_i \in \text{QS} \mid e_i \text{ is type-2 and b-value}(e_{i+1}) = \alpha\}.$$

此外，对于任意 $e_i \in X_\alpha$，由于 e_i 是一个 2 型元素，b 值(e_i) $\leqslant$ b 值(e_{i+1}) $= \alpha$。我们可以利用这一点和命题 3.5 来界定每个 X_α 的大小，如下所示：

$$|X_\alpha| \cdot (2^{\alpha-1} - 2) \leqslant \sum_{e_i \in X_\alpha} g_i + g_{i+1} \leqslant 2 \sum_{e_j \in \mathtt{QS} \cap \mathrm{Band}_{\leqslant \alpha}} g_j, \tag{7}$$

其中第二个不等式成立，是因为如上所述，e_i 和 e_{i+1} 都属于带$\leqslant\alpha$。因此，证明的最后一步是对 g中元素的分位数摘要 $\cap$ 带$\leqslant\alpha$的所有 g值之和进行界定。这很容易做到，因为$\leqslant\alpha$ 值跟踪被覆盖元素的数量，而带$\leqslant\alpha$ 中的元素根据算法的构造（删除步骤中的规则 (1)）以及观察 2.5，只能覆盖带$\leqslant\alpha$ 中的元素；再结合公式 (2)中对带大小的上界，就可以证明以下命题。

命题 3.6。在时间 t 执行删除步骤后，对任意 α，$\sum_{e_i \in \mathtt{QS} \cap \text{带}_{\leqslant \alpha}} g_i \leqslant \ell \cdot 2^{\alpha+1}$.

证明。固定一个 e_i 位于分位数摘要 $\cap$带$\leqslant\alpha$ 中的元素，并回顾 $C(e_i$中对被覆盖元素的定义，见第 2.3 节。在算法 1中，只有当删除时 b 值(e_{i-1}) 6 b 值(e_i) 时，我们才可以删除元素。根据观察 2.5，这一条件此后始终成立，这意味着所有 $C(e_i$都属于带$\leqslant\alpha$（其中 α 是 e_i当前的带值）。于是我们有：

$$\begin{aligned}
\sum_{e_i \in \mathtt{QS} \cap \mathrm{Band}_{\leqslant \alpha}} g_i &= \sum_{e_i \in \mathtt{QS} \cap \mathrm{Band}_{\leqslant \alpha}} |C(e_i)| && (\text{as } g_i = |C(e_i)| \text{ by Eq (5)}) \\
&= \Big| \bigcup_{e_i \in \mathtt{QS} \cap \mathrm{Band}_{\leqslant \alpha}} C(e_i) \Big| && (\text{由于 } C(e_i) \text{ 的 } e_i \text{ 中 分位数摘要 两两不相交，根据 方程 (5)}) \\
&\leqslant |\mathrm{Band}_{\leqslant \alpha}| && (\text{因为 } C(e_i) \text{ 的元素属于带}\leqslant\alpha) \\
&\leqslant \ell \cdot 2^{\alpha+1}, && (\text{by the bound in Eq (2)})
\end{aligned}$$

完成证明。 ∎ 命题 3.6

将 命题 3.6 中的界代入 公式 (7)，可得：

$$|X_\alpha| \cdot (2^{\alpha-1} - 2) \leqslant \ell \cdot 2^{\alpha+2},$$

这进而意味着 $|X_\alpha| = O(\ell)$，当 $\alpha > 2$ （并且对于 $\alpha \leqslant 2$ 元素总数至多为$8 \cdot \ell = O(\ell)$ 时，根据 公式 (2) 无论如何都成立）。考虑到集合 $O(\log t)$ 只有 X_α 种选择（因为不同带的数量为 $O(\log t)$，根据 公式 (2)），我们得到最终结果。 ∎

引理 3.3 和 3.4 表明，在时间 t执行删除步骤后，分位数摘要 的大小为 $O(\ell \cdot \log^2 t)$。这意味着摘要的总大小为 $O(\frac{1}{\varepsilon} \cdot \log^2(\varepsilon n))$，并据此完成 引理 3.2的证明，因为 $t = O(\varepsilon n)$ 且 $\ell = O(1/\varepsilon)$（定义 2.2）。这里有一条重要的注释。

注 3.7. 在空间分析中，我们将删除步骤后摘要中类型 2 元素的数量界定为 $O(\ell \cdot \log t) = O((1/\varepsilon) \cdot \log(\varepsilon n))$，这本身已经相当高效。然而，如图 3 所示，在最坏情况下，每个类型 2 元素对应的 1 型元素数量可能达到 $O(\log t)$。因此，算法 1 最终可能会在摘要中存储多达 $O(\ell \cdot \log^2 t) = O((1/\varepsilon) \cdot \log^2(\varepsilon n))$ 个 1 型元素，导致其空间需求并非最优。

正如我们在 注 3.7 中所说，算法 1 的次优性来源之一，是摘要中存储的 1 型元素数量远多于 2 型元素。改进这一点的一种方法是主动尝试减少存储的 1 型元素数量。粗略地说，只有在删除摘要中的 2 型元素不会导致形成一长串 1 型元素时，才删除该元素（例如，图 3 所示）。请注意，我们的算法 3 正是这样做的：只有当某个元素所在的整个段也能一并删除时，才删除该元素。

我们以如下说明结束本节；这将有助于我们高效地实现该算法。

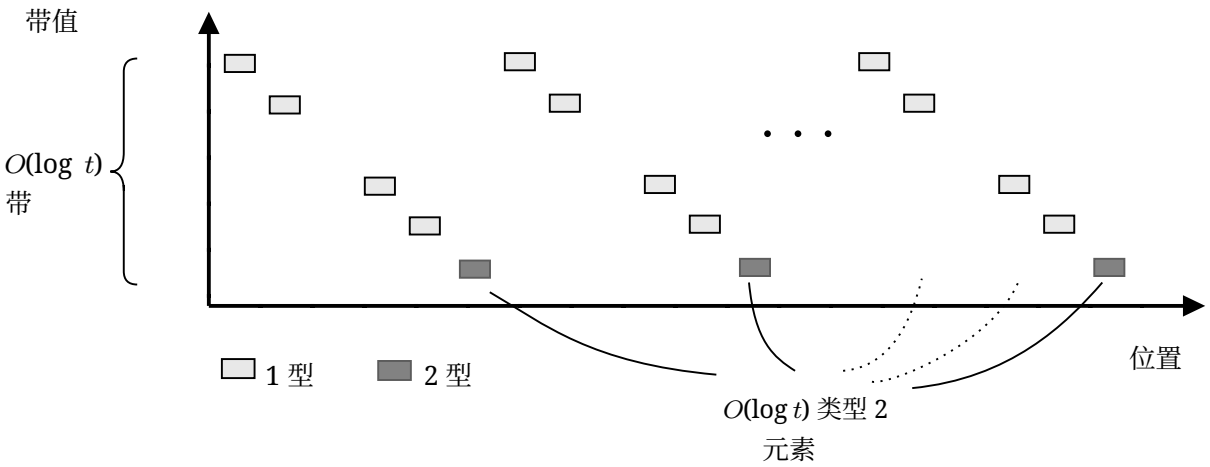


图 3: 图中的每个方块代表存储在 分位数摘要中的一个元素。元素的秩沿水平方向递增。该图说明了为什么 算法 1 最终可能会在 $O(\ell \log^2 t)$ 个元素的空间中存储于 分位数摘要中。根据 引理 3.4，分位数摘要中最多可能有 $O(\ell \cdot \log t)$ 个 2 型元素。这些 2 型元素中的每一个前面都可能有一串 $O(\log t)$ 个 1 型元素（因为有 $O(\log t)$ 条带）。

注 3.8 (延迟删除)。 假设在 算法 1 中，我们不在每个数据块之后执行 删除步骤（第 (ii) 行），而是在插入 k 个连续数据块后再执行这些步骤，其中 $k > 1$为某个整数；那么，该算法的空间复杂度仅增加一个 $O(k \cdot \ell)$ 加性项。

请注意，这仅仅是因为在任意时刻 t 执行删除步骤后，元素数量会减少到 $O(\ell \cdot \log^2 t)$，如前所证。因此，额外的空间仅用于在分位数摘要中存储额外的 $O(k \cdot \ell)$ 个已插入但尚未删除的元素。

3.2 算法 1的高效实现

本节将介绍 算法 1的一种高效实现。它与 算法 3 在 第 6.2 节中介绍的实现非常相似。形式化地，我们证明如下结论：

引理 3.9。 对于任意 $\varepsilon > 0$ 以及长度为 n的数据流，都存在一种 算法 1 实现，其$O\big(\log(1/\varepsilon) + \log\log(\varepsilon n)\big)$每个元素的最坏情况处理时间。

第一部分：存储分位数摘要：我们将分位数摘要存储为一棵平衡二叉搜索树（二叉搜索树），其中每个节点都包含分位数摘要中的一个元素及其元数据。对于每个元素 e，我们存储 $g(e)$ $\Delta(e)$ 和 $t_0(e)$。二叉搜索树的排序键是元素的值。插入和删除操作分别向二叉搜索树中插入和删除元素。

第二部分：执行删除步骤：我们使用以下算法高效地执行删除步骤（所需时间与摘要大小成正比）。

> 算法：高效执行删除步骤：
>
> 1. 对 分位数摘要 （它是一棵二叉搜索树）进行中序遍历，以获得一个按值排序的临时（双向链表）元素列表。2. 使用 观察 2.7 计算分位数摘要中所有元素的 b 值。
>
> 3. 从较大的元素开始，按降序遍历列表。对于每个元素 e_i，如果它同时满足 算法 1 中提到的两个删除条件，则将其从二叉搜索树（以及列表）中删除。

首先，我们给出一种具有较快摊销更新时间的算法，然后利用标准技术展示如何实现该算法，使其最坏情况更新时间也具有相同的渐近复杂度。

实现 1. 算法 1 的高效实现算法 1

- 将分位数摘要初始化为空的平衡二叉搜索树。
- 删除时间 $\leftarrow 2$。
- 对于每个时间步 t，当有数据项到达时（$x_{(t)\,1}$，...，$x_{(t)\,\ell}$）：（i）对数据块中的每个元素运行插入($x_{(t)\,j}$)。（ii）如果（t =删除时间）：– 执行删除步骤并更新删除时间 $\leftarrow$ 删除时间 $+\lceil \log^2 t \rceil$。

空间分析。实现 1 的空间复杂度为 $O(\frac{1}{\varepsilon}\log^2(\varepsilon n))$。这仅仅是因为，在任意时间 t的删除步骤之后，我们会将删除延迟 $\lceil \log^2 t \rceil$ 个时间步。根据注 3.8，这意味着所用空间只会增加一个 $O(\ell \log^2 t) = O(\frac{1}{\varepsilon}\log^2(\varepsilon n))$ 的加性项，因为 $t = O(\varepsilon n)$ 和 $\ell = O(1/\varepsilon)$（定义 2.2）。

时间分析。由于分位数摘要以二叉搜索树的形式存储，对分位数摘要执行插入或删除操作所需的时间为$O(\log s)$，其中 s是分位数摘要的大小。根据上文的空间分析，$s = O((1/\varepsilon)\cdot\log^2(\varepsilon n))$。由此得到以下观察：

观察 3.10。对于长度为 n的数据流，实现 1 执行所有插入 和 删除 操作所需的总时间为 $O(n\cdot(\log(1/\varepsilon) + \mathrm{loglog}(\varepsilon n)))$。

该观察基于以下事实：每个元素在分位数摘要中至多插入和删除一次，且每次插入或删除所需的时间为 $O(\log s) = O\big(\log(1/\varepsilon) + \mathrm{loglog}(\varepsilon n)\big)$。实现实现 1所需的、未计入观察 3.10 的唯一时间，是用于确定要删除哪些元素的部分，我们将在下文对此给出上界。

命题 3.11。对于长度为 n的数据流，实现 1 在所有已执行的删除步骤中，确定要删除哪些元素所需的总时间为 $O(n)$。

证明。在一次删除步骤中，判断需要删除哪些元素所需的时间为 $O(s) = O(\frac{1}{\varepsilon}\log^2(\varepsilon n))$. 这是因为：创建一个链表，然后计算所有元素的 b 值，这些操作可在 $O(s)$ 时间内完成。接着，从最大元素到最小元素线性遍历链表（以检查删除条件是否成立），需要 $O(s)$ 时间。

因此，我们现在着重界定整个数据流中可能执行的删除步骤数量。形式化地，

命题 3.12. 对于长度为 n 的数据流，实现 1 执行的删除步骤数量为$O(\varepsilon n/\log^2(\varepsilon n))$。

证明。我们将所有时间步划分为长度为 $[2^i, 2^{i+1})$ 的区间，其中 $1 \leqslant i \leqslant \lceil \log(\varepsilon n) \rceil$。由于在时间步 t 执行删除步骤后，我们会等待 $\lceil \log^2 t \rceil$ 个时间步，因此，在区间 $[2^i, 2^{i+1})$ 内发生的任意两个连续删除步骤之间至少相隔 i^2 个时间步。鉴于该区间的长度为 2^i，我们在区间 $[2^i, 2^{i+1})$ 内执行删除步骤的次数至多为 $1 + \frac{2^i}{i^2}$。因此，

$$\#\ \text{deletion steps performed} \leqslant \sum_{i=1}^{\lceil \log(\varepsilon n)\rceil} 1 + \frac{2^i}{i^2} = O\left(\frac{\varepsilon n}{\log^2(\varepsilon n)}\right). \quad \blacksquare \text{ 命题 3.12}$$

根据命题 3.12，我们执行 $O\big(\frac{\varepsilon n}{2\log(\varepsilon n)}\big)$ 个删除步骤；在每个步骤中，我们花费 $O\big(\frac{1}{\varepsilon}\log^2(\varepsilon n)\big)$ 时间来确定需要删除哪些元素，由此引理得证。

$\blacksquare$

观察 3.10 和 命题 3.11 显然意味着，实现 1 在长度为 n 的数据流上所花费的总时间为 $O\big(n \cdot (\log(1/\varepsilon) + \log\log(\varepsilon n)\big)$。因此，每个元素的摊销更新时间为 $O\big(\log(1/\varepsilon) + \log\log(\varepsilon n)\big)$。

最坏情况更新时间。现在我们来考察任意元素的最坏情况更新时间。在最坏情况下，执行删除步骤时，我们可能需要从 $O(s$个元素中删除分位数摘要 中的元素，这需要$O(s\log\ s)$ 时间。为了降低最坏情况更新时间，我们对实现 1做了一点小修改。我们注意到，下一次调用删除步骤是在经过 $\lceil \log^2 t \rceil$ 个时间步之后。为了降低最坏情况时间，我们将执行删除步骤所需的时间分摊到下一次删除步骤之前的所有元素上。形式化地说，有如下结论：

断言 3.13。存在一种算法 1 的实现，其最坏情况更新时间为 $O(\log\ \frac{1}{\varepsilon} + \log\log(\varepsilon n))$（每个元素），且不会影响其渐近空间复杂度。

证明。由于分位数摘要是二叉搜索树，因此插入元素可在 $O(\log s)$ 时间内完成。所以，唯一耗时超过 $O(\log s)$ 的步骤是执行删除步骤。设 t 为执行删除步骤的一个时间步。下一次删除步骤发生在时间步 $t + \lceil \log^2 t \rceil$。因此，我们将删除步骤均匀分摊到 $\lceil \log^2 t \rceil / 2$ 个时间步中，并将数据流中到达的元素存储在缓冲区 B 中。每个元素的更新时间为 $O(\log s)$，因为有 $O(s)$ 个元素被插入到 B 中，而删除步骤需要 $O(s\log\ s)$ 时间。注意，我们将插入操作均匀分摊，因此每个元素的最坏情况更新时间为 $O(\log s)$。在剩余的 $\lceil \log^2 t \rceil / 2$ 个时间步中，我们每个时间步插入两个元素，每个元素的最坏情况更新时间为 $O(\log s)$。此外，在时间 $t + \lceil \log^2 t \rceil$，缓冲区 B 为空，因为有$2 \cdot \lceil \log^2 t \rceil / 2 = \lceil \log^2 t \rceil$ 个元素被插入分位数摘要中。因此，任意元素的最坏情况更新时间为 $O(\log\ s) = O(\log \frac{1}{\varepsilon} + \log\log\ \varepsilon n)$。此外， B的大小为 $O(s)$，因此空间仅增加一个常数因子。 ∎

至此，引理 3.9的证明完成。现在我们来说明，回答分位数查询所需的时间为 $O(\log s)$。

命题 3.14。 经过实现 1创建的摘要，在使用 $O(s)$ 预处理时间后，可以回答一系列分位数查询，每次查询所需 $O(\log s)$ 时间，其中 s 是摘要的大小。

证明。 根据公式 (4)，可通过一次中序遍历计算分位数摘要中所有元素的 r-最小值和 r-最大值。这是预处理步骤，所需时间为 $O(s)$。要回答一个 ϕ-分位数查询，只需找到一个元素，使其 r-最小值和 r-最大值都落在区间 $[(\phi - \varepsilon)n,\ (\phi + \varepsilon)n]$。根据命题 2.1 ，这样的元素必定存在；由于 $O(\log s)$ 时间，因为分位数摘要按元素的值排序并以二叉搜索树的形式存储，因此也按 r-最小值排序。 ∎

命题 3.14 结合引理 3.2 和引理 3.9 证明了定理 1 ，因为 $s = O(\frac{1}{\varepsilon} \cdot \log^2(\varepsilon n))$。

4 最终的 $O(\frac{1}{\varepsilon} \cdot \log\ (\varepsilon n))$ 大小摘要

本节将介绍 GK 摘要。正如我们在 注 3.7 中所讨论的，在该算法中，我们会主动避免形成由大量类型 1 元素构成的长序列（见 图 3）。粗略而言，只要删除某个元素及其整个段不会违反不变量 1（以及相同的关于 b 值的条件），我们就删除该元素及其整个段。令 g_i^* 表示 g 中 seg(e_i 内各元素的值之和。下面给出该算法的正式描述。

算法 2：一种改进的分位数摘要更新算法。

对于每个时间步 t，当有数据项到达时 $(x_1^{(t)}, \ldots, x_\ell^{(t)})$：

(*i*) 对数据块中的每个元素运行插入($x_j^{(t)}$)。

(*ii*) 当 QS 中存在满足以下条件的元素 e_i 时：

$$(1)\ \text{b-value}(e_i) \leqslant \text{b-value}(e_{i+1}) \qquad \underline{\text{and}} \qquad (2)\ g_i^* + g_{i+1} + \Delta_{i+1} \leqslant t$$

对 $\{e_i\} \cup$ 段(e_i)中的 e_k 运行删除(e_k)。

算法 1 与算法 2 的主要区别在于，后者在尝试删除元素及其所在段时，将它们视为“一个整体”，也就是说，要么连同整个段一起删除该元素，要么完全不删除该元素（请注意，该段中的元素也可能被单独删除）。我们要指出，尽管我们对算法 2 的描述与 GK 摘要的呈现方式 [GK01], 有所不同，但这两种算法的行为几乎完全相同。

定理 2。对于任意 $\varepsilon > 0$ 以及长度为 n的数据流，算法 2 维护一个 ε-近似分位数摘要，占用 $O(\frac{1}{\varepsilon} \cdot \log(\varepsilon n))$ 空间。此外，存在一种算法 2 的实现，对每个元素的最坏情况更新时间为 $O(\log(\frac{1}{\varepsilon}) + \text{loglog}(\varepsilon n))$ 。

算法 2 维持 不变量 1 因为它只有在满足以下条件时才会删除一个元素 e_i （以及 seg(e_i)）： $g_i^* + g_{i+1} + \Delta_{i+1} \leqslant t$；根据 观察 2.8，这意味着 $g_{i+1} + \Delta_{i+1} \leqslant t$ 在删除后（所有其他（g， Δ）值保持不变）。如第 2.3 节所述，维持 不变量 1 直接意味着分位数摘要 在整个数据流中始终是一个 ε-近似分位数摘要。因此，我们在分位数摘要 下，重点分析该算法的大小上界，见第 4.1 节；并在第 4.2 节 给出其高效实现，以证明定理 2。

4.1 空间分析

在本小节中 ，我们证明 算法 2所使用空间的上界。形式化地，我们有如下 ：

引理 4.1。 对于任意 $\varepsilon > 0$ 以及长度为 n的数据流，算法 2 在 ε-近似分位数摘要中维持 $O(\frac{1}{\varepsilon} \cdot \log(\varepsilon n))$ 空间。

回想一下，算法 1 的一个简单性质是，带$_{\leqslant\alpha}$ 中的元素只能覆盖带$_{\leqslant\alpha}$ 中的元素。我们认为，算法 2中这一性质依然成立。

观察 4.2。在任何时刻，分位数摘要中带$_{\leqslant\alpha}$ 的元素只会覆盖带$_{\leqslant\alpha}$ 中的元素。

这是因为：当 e_i 和 seg(e_i被 e_{i+1}覆盖时，算法 2 会确保 b 值(e_i) $\leqslant$b 值(e_{i+1})。根据定义 6.1，seg(e_i) 包含 b 值小于 b 值(e_i)的元素。因此，$C(e_{i+1})$包含 b 值至多为 b 值(e_{i+1})的元素；根据观察 2.5，在之后的时刻这一性质仍然成立。

做出上述观察后，我们现在来证明分位数摘要大小的界。我们对第 3.1 节中 1 型和 2 型元素的定义稍作修改并沿用。时间 t的删除步骤之后，元素 e_i 在分位数摘要中称为1 型元素，当且仅当它满足条件 b 值(e_i) >b 值(e_{i+1})；若它仅满足条件 $g_i^* + g_{i+1} + \Delta_{i+1} > t$，则称为2 型元素。删除步骤之后，分位数摘要中的任一元素都满足第一个或第二个条件（最后一个元素 $e_s = +\infty$除外）；否则算法 2 就会删除该元素。

我们首先对类型 1 元素的数量进行界定。与算法 1不同，此处我们可以证明，在任意时间 $O(\ell \cdot \log t)$ 内，类型 1 元素的数量为 t，比之前少一个因子 $O(\log t)$；这正是改进算法 2空间复杂度的关键。下面的引理是证明的核心。

引理 4。 3. 在时间 t 执行删除步骤后，分位数摘要中的类型 1 元素数量为 $O(\ell \cdot \log t)$.

证明。令我们部分 将类型 1 元素划分为 $B^{(t)}$ 个集合 Y_0、Y_1、……、$Y_{B^{(t)}-1}$，其中对于任意带值 α ：

$$Y_\alpha := \{e_i \in \mathtt{QS} \mid e_i \text{ is type-1 and b-value}(e_{i+1}) = \alpha\};$$

（注意，Y_α 中的元素满足：其后一个元素的带值为 α，而不是它们自身的带值；参见脚注3）。这种划分方式类似于引理 3.4 中的划分，但现在针对的是 1 型元素。我们将证明任意集合 Y_α 的大小为 $O(\ell)$。为此，我们将 e_i 中每个元素 Y_α 映射到 $e_j > e_i$中 b 值大于 α的第一个元素；示意图见图 6a 。令 T_α 为所有此类元素 e_j构成的集合。对于任意 e_j 中的 T_α，我们可以说：

$$g^*_{j-1} + g_j + \Delta_j > t, \tag{8}$$

因为 e_j 是 b 值大于 α 的第一个元素（因此 e_{j-1} 是一个 2 型元素）。容易看出，Y_α 中的任意两个元素都不会映射到 T_α中的同一个元素。因此，$|Y_\alpha| = |T_\alpha|$，所以只需给 $|T_\alpha|$一个上界即可。

由于 b 值(e_j) 大于 b 值(e_{i+1}) $= \alpha$，观察 2.5 保证 e_j 在时间 $t' = t_0(e_{i+1})$ 时存在于分位数摘要中。令 g'_j 为 e_j 在 t'时的 g-值。由于 Δ_j 不是时间的函数，时间 t' 时的不变量 1 推出：

$$g'_j + \Delta_j \leqslant t'. \tag{9}$$

将 方程（8）减去方程（9），并利用观察 2.4 给出的界，可得：

$$g^*_{j-1} + (g_j - g'_j) > t - t' \geqslant 2^{\alpha-1} - 2. \tag{10}$$

在上面的方程中：

(i) 项 $(g_j - g'_j)$ 统计了时间 t' 之后，依据 e_j 方程（5），被 e_j 覆盖的元素数。我们将在命题 4.4中证明，在时间 t'，e_j 只覆盖带$\leqslant\alpha$中的元素。

(ii) 项 g^*_{j-1} 统计了被 e_{j-1} 和 seg(e_{j-1}) 覆盖的元素。根据定义 6.1，e_{j-1} 以及 seg(e_{j-1}) 中的所有元素的 b 值均为 $\leqslant \alpha$。观察 4.2 使我们可以得出结论，g^*_{j-1} 统计的元素也位于带$\leqslant\alpha$ 中。

（iii）最后，根据式（5），对于 e_{j_1} 和 e_{j_2} 中不同的 T_α，由 e_{j_1-1} 及其段覆盖的元素集合，与由 e_{j_2-1} 及其段覆盖的元素集合互不相交（如图6所示）。

这意味着式（10）左端对所有 T_α求和后，与带$\leqslant\alpha$ 中元素的总数成正比，即 $O(\ell \cdot 2^\alpha)$，根据公式 (2)。形式化地，

$$|T_\alpha| \cdot (2^{\alpha-1} - 2) \leqslant \sum_{e_j \in T_\alpha} g^*_{j-1} + \sum_{e_j \in T_\alpha} (g_j - g'_j) \leqslant |\mathrm{Band}_{\leqslant\alpha}| + |\mathrm{Band}_{\leqslant\alpha}| \leqslant \ell \cdot 2^{\alpha+2}. \tag{11}$$

这就说明了所需的 $|T_\alpha| = |Y_\alpha| = O(\ell)$。

命题 4.4. 在时间 t，t' 之后由 e_j 覆盖的所有元素，其 b 值至多为 α。[2]

证明。假设存在一个 b 值为 $> \alpha$ 的元素，它在时间 t'之后被 e_j 覆盖。所有此类元素都小于 e_j 且大于 e_{i+1}，因为它们是被 e_j覆盖的。现在考虑其中最小的元素 $e_\circ$。显然，e_{i+1} 在 t'时属于 e 的段。由于 e 在时间 t 不属于摘要，因此它必定已被删除。这意味着包含 e_{i+1}的段已被删除。这说明 e_{i+1} 也被删除了，矛盾。命题 4.4 ∎

我们证明了，对于任意 α，$|Y_\alpha| = O(\ell)$。由于 $O(\log t)$ 有 α (公式 (2)) 种可能的取值，因此共有$O(\ell \cdot \log t)$ 个类型 1 元素，由此完成 引理 4.3 的证明。 ∎

现在我们将界定分位数摘要中类型 2 元素的数量。分析过程与引理 3.4 的证明几乎完全相同，这里给出该分析以确保完整性。

[2]注意，命题 4.4 对 算法 1 并不成立，因此我们无法对其进行类似的分析。这里空间界更优的原因在于，删除元素的同时也删除了该段。

引理 4。 5. 在时间 t 执行删除步骤后，分位数摘要中的类型 2 元素数量为 $O(\ell \cdot \log t)$.

证明。 任意一个 2 型元素 e_i 在 分位数摘要中都具有如下性质： $g_i^* + g_{i+1} + \Delta_i > t$。我们将利用这一事实得出结论：元素 e_i、其所在段以及 e_{i+1} 覆盖了“足够多”的元素。下一个断言是算法 1中命题 3.5的类似形式。

命题 4.6。在时间 t 执行删除步骤后，对于任意 2 型元素 e_i, $g_i^* + g_{i+1} \geqslant 2^{\text{b-value}(e_{i+1})-1} - 2$。

证明。 由于 e_i 是一个 2 型元素， $g_i^* + g_{i+1} + \Delta_{i+1} > t$。根据 公式 (6)， $\Delta_{i+1} \leqslant t_0(e_{i+1})$，因此，

$$g_i^* + g_{i+1} > t - t_0(e_{i+1}) \geqslant 2^{\text{b-value}(e_{i+1})-1} - 2,$$

其中第二个不等式由 观察 2.4得出。 ∎ 命题 4.6

断言 4.6 给出了每个 2 型元素 g^*的下界 e_i ，该下界是 g的值和下一个元素 e_{i+1}的带值的函数。因此，我们将 2 型元素划分为 $B^{(t)}$ （可能带的数量）个集合 X_0， X_1， ....., $X_{B^{(t)}}$ 其中，对于任意带值 α:

$$X_\alpha := \{e_i \in \texttt{QS} \mid e_i \text{ is type-2 and b-value}(e_{i+1}) = \alpha\}.$$

此外，对于任意 $e_i \in X_\alpha$，由于 e_i 是一个 2 型元素，b-value(e_i) $\leqslant$ b-value(e_{i+1}) $= \alpha$。我们将断言 4.6 中的不等式对 X_α 中的每个元素求和，得到:

$$|X_\alpha| \cdot (2^{\alpha-1} - 2) \leqslant \sum_{e_i \in X_\alpha} g_i^* + g_{i+1}. \tag{12}$$

现在，我们尝试用 α 的某个函数给上式中的求和找一个上界。

命题 4.7。 在时间 t的删除步骤之后，对于任意 $\alpha \geqslant 0$， $\sum_{e_i \in X_\alpha} g_i^* \leqslant 2 \sum_{e_j \in \texttt{QS} \cap \text{带}_{\leqslant \alpha}} g_j$.

证明。由于 X_α 中元素的线段可能彼此重叠，因此无法直接对这个求和进行界定。为了解决这个问题，我们将 X_α 划分为两个不相交的集合 $X_\alpha \cap \text{带}_\alpha$ 和 $X_\alpha \cap \text{带}_{\leqslant \alpha-1}$，使得每个集合中任意两个元素的线段互不相交。

$X_\alpha \cap \text{带}_\alpha$ 中任意两个不同元素的线段互不相交。这是因为它们的线段（其中包含来自小于 α 的带的元素）之间至少隔着一个来自带$_\alpha$ 的元素。因此，

$$\sum_{e_i \in X_\alpha \cap \text{Band}_\alpha} g_i^* \leqslant \sum_{e_j \in \texttt{QS} \cap \text{Band}_{\leqslant \alpha}} g_j.$$

根据 X_α 的定义， $X_\alpha \cap \text{带}_{\leqslant \alpha-1}$ 中任意两个不同元素之间至少有一个来自带$_\alpha$ 的元素。因此， $X_\alpha \cap \text{带}_{\leqslant \alpha-1}$ 中的元素所对应的、包含来自带$\leqslant \alpha$ 的元素的线段也互不相交。因此，

$$\sum_{e_i \in X_\alpha \cap \text{Band}_{\leqslant \alpha-1}} g_i^* \leqslant \sum_{e_j \in \texttt{QS} \cap \text{Band}_{\leqslant \alpha}} g_j,$$

由此可得命题成立。 ∎ 命题 4.7

下面的命题对 g 的值之和进行界定，其中求和对象是 分位数摘要中来自带$_{\leqslant \alpha}$ 的元素，界为 $\ell \cdot 2^{\alpha+1}$。其证明与命题 3.6完全相同，故略去。

命题 4.8。 在任意时间 t 且对于任意 α， $\sum_{e_i \in \texttt{QS} \cap \text{带}_{\leqslant \alpha}} g_i \leqslant \ell \cdot 2^{\alpha+1}$.

将 命题 4.7 和 命题 4.8 中的界代入 公式 (12)，我们得到:

$$|X_\alpha| \cdot (2^{\alpha-1} - 2) \leqslant \sum_{e_i \in X_\alpha} g_i^* + \sum_{e_j \in \texttt{QS} \cap \text{Band}_{\leqslant \alpha}} g_j \leqslant 3 \sum_{e_j \in \texttt{QS} \cap \text{Band}_{\leqslant \alpha}} g_j \leqslant 3 \cdot \ell \cdot 2^{\alpha+1},$$

这意味着 $|X_\alpha| = O(\ell)$。由于根据 $O(\log t)$ 个集合 X_α，而根据 $O(\log t)$ 个带来自 公式 (2)，因此引理得证。 ∎

现在我们已经证明，在时间 t执行删除步骤后，分位数摘要 中的 1 型元素数量为 $O(\ell \cdot \log t)$，由 引理 4.3 可知，分位数摘要中的 2 型元素数量为 分位数摘要 中的 $O(\ell \cdot \log t)$，由 引理 4.5 可知。由于分位数摘要中的每个元素（除 $+\infty$外）要么是 1 型，要么是 2 型，因此分位数摘要中的元素总数为$O(\ell \cdot \log t)$。

至此，引理 4.1 的证明完成， $t = O(\varepsilon n)$ 和 $\ell = O(\frac{1}{\varepsilon})$（定义 2.2）。我们用以下备注结束对空间复杂度的讨论；这与算法 1的注 3.8类似。

备注 4.9 （延迟删除）。 假设在算法 2中，我们不在每个数据块之后执行删除步骤（即第 (ii) 行），而是在插入 k 个连续数据块后再执行， $k > 1$其中 $O(k \cdot \ell)$ 为某个整数；那么，该算法的空间复杂度只会增加一个

论证与注 3.8中的相同；在任意时刻 t执行删除步骤后，元素数量会减少至 $O(\ell \cdot \log t)$，如前所证。因此，额外空间仅用于存储插入 $O(k \cdot \ell)$ 个额外元素分位数摘要中。

4.2 算法 2的高效实现

我们采用与第 3.2 节类似的策略来高效地实现算法 2。具体而言，我们证明如下结论：

引理 4.10。对于任意 $\varepsilon > 0$ 以及长度为 n的数据流，存在一种算法 2的实现，其$O(\log(\frac{1}{\varepsilon}) + \log\log(\varepsilon n))$ 最坏情况更新时间为每个元素。

第一部分：存储分位数摘要： 摘要分位数摘要以平衡二叉搜索树的形式存储，与实现中完全相同 1.

第二部分：执行删除步骤：删除步骤的执行方式也与算法 1的快速实现非常相似。唯一的变化是，在决定删除哪些元素时，会检查算法 2的删除条件。不过，这需要计算摘要中元素的 g^* 值，其计算方式与 G^* 值类似，如第 6.2 节所示。现在我们给出算法 2的一种实现，使每个元素的摊销更新时间较快。利用第 6.2 节中描述的技术，可以证明每个元素的最坏情况更新时间也具有相同的界。

实现 2. 算法 2 的高效实现

- 将分位数摘要初始化为空的平衡二叉搜索树。
- 删除时间 $\leftarrow 2$。
- 对于每个时间步 t，当有数据项到达时（$x_{(t)\,1}, \ldots, x_{(t)\,\ell}$）：（$i$）对数据块中的每个元素运行插入（$x_{(t)\,j}$）。（$ii$）如果（$t$ =删除时间）：– 执行删除步骤并更新删除时间 $\leftarrow$ 删除时间 $+ \lceil \log t \rceil$。

空间分析 快速实现的空间复杂度仍为 $O(\frac{1}{\varepsilon}\log(\varepsilon n))$。这是因为在任意时间 t执行删除步骤后，我们会将删除延迟 $\lceil \log t \rceil$ 个时间步。因此，根据 备注 4.9，空间用量只会增加一个大小为 $O(\ell \log t) = O(\frac{1}{\varepsilon}\log(\varepsilon n))$ 的加性项，因为 $t = O(\varepsilon n)$ 和 $\ell = O(1/\varepsilon)$（定义 2.2）。

时间分析 用于界定所述实现的时间复杂度的论据与 第 3.2 节 中的论据非常相似。

我们利用分位数摘要是一个含有 $O(\frac{1}{\varepsilon}\log(\varepsilon n)$ 个元素的二叉搜索树这一事实，得出如下观察。

观察 4.11。 对于长度为 n的数据流，实现 2 执行所有插入 和 删除 操作所需的总时间为 $O(n \cdot (\log(1/\varepsilon) + \log\log(\varepsilon n)))$。此外，这也是实现 2的总运行时间，但不包括在删除步骤中决定要删除哪些元素所需的时间。

接下来，我们对所有删除步骤中用于决定删除哪些元素的时间进行界定。这样就完成了我们的分析。

命题 4.12. 对于长度为 n 的数据流，实现 2 在所有已执行的删除步骤中，确定需要删除哪些元素所花费的总时间为 $O(n)$。

证明。容易验证，在一个删除步骤中，确定需要删除哪些元素所花费的时间为 $O(s) = O(\frac{1}{\varepsilon}\log(\varepsilon n))$。因此，只需界定整个数据流中可能需要执行的删除步骤数。形式化地，

命题 4.13. 对于长度为 n 的数据流，实现 2 执行的删除步骤数为$O\Big(\varepsilon n/\log(\varepsilon n)\Big)$。

该命题的证明从略，因为它与 命题 3.12 的证明非常相似。 ∎ 命题 4.12

观察 4.11 和 命题 4.12 表明，实现 2 处理长度为 n 的数据流所花费的总时间为 $O\Big(n\cdot\big(\log(1/\varepsilon)+\log\log(\varepsilon n)\big)\Big)$。因此，每个元素的摊销更新时间为 $O\big(\log(1/\varepsilon)+\log\log(\varepsilon n)\big)$。可以通过将删除操作分散到多个步骤中来修改此实现，从而降低每个元素处理时间的最坏情况。形式化地，我们有：

命题 4.14。 存在一种 算法 2 实现，其最坏情况更新时间为 O（log $\frac{1}{\varepsilon}+\log\log(\varepsilon n)$）每个元素，且不会影响其渐近空间复杂度。

该策略是将一个删除步骤中的删除分散到下一个删除步骤之前发生的各个时间步中。至此， 引理 4.10的证明完成。

5 基本设置（加权设置）

现在，我们介绍加权设置下分位数摘要的基本设置和预备定义。首先，我们将定义 1.2 中的问题等价地改写为无权分位数问题，并先定义展开流的概念。定义 1.2 展开 数据流。

展开流。对于加权数据流 S_w，我们将其对应的展开流 Unfold(S_w) 定义为包含 $w(x_i)$ 个 x_i 副本的流，其中 $1\leqslant i\leqslant n$。更明确地说，

$$\mathsf{Unfold}(S_w) := \langle x_1^{(1)}, x_1^{(2)}, \ldots, x_1^{(w(x_1))}, \ldots x_n^{(1)}, x_n^{(2)}, \ldots, x_n^{(w(x_n))}\rangle$$

其中 $x_i^{(j)}$ 是元素 x_i 的第 j 个副本。

很容易验证，问题的目标（如定义 1.2所述）等价于为 ε构造一个近似分位数摘要：Unfold(S_w)。请注意，为相同元素赋秩时，为了打破平局，我们假设在Unfold(S_w)中较早出现的元素具有较低的秩。顺带指出，尽管我们提出的算法不会“展开”数据流，但在分析该算法时，我们仍将使用Unfold(S_w)。

我们用 WQS 表示算法为 S_w 创建的摘要。WQS 由数据流中的一部分元素以及关于所存储元素的一些辅助元数据组成。我们用 e_i 表示 WQS 中存储的数据流里的第 i 大元素。我们用 $e_i^{(j)}$ 表示 Unfold(S_w) 中 e_i 的第 j 个副本，其中 $1\leqslant j\leqslant w(e_i)$。我们还用 e 表示摘要中的任意元素（当秩无关紧要时）。WQS 中存储的数据流元素数量记为 s。对于每个元素 e，WQS 存储 $w(e)$。我们为每个元素 e 存储的另一项主要信息是其 r-min 和 r-max 值，现定义如下：

- r-最小值(e) 和 r-最大值(e)：WQS 对 $e^{(1)}$（Unfold(S_w) 中首次出现的 e 副本）的秩所维护的上下界。由于我们没有存储所有元素，因此无法确定某个已存储元素的确切秩，所以重点在于维护恰当的上下界。

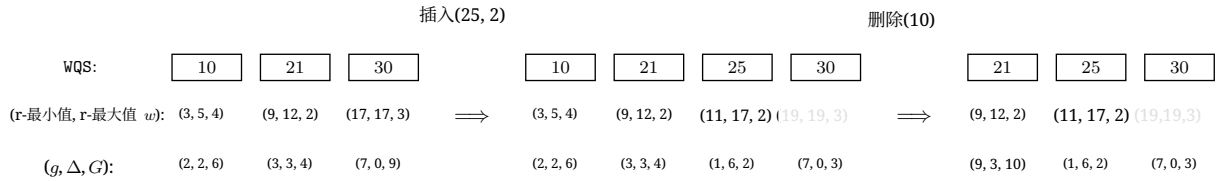


图4: 从某个任意状态开始，摘要中的更新操作示意图（本图中的参数 (g, Δ, G) 定义于 第 5.2 节）。

为处理后续出现的边界情况，我们假设 WQS 中包含一个哨兵元素 e_0，并定义 r-最小值(e_0) =r-最大值(e_0) $= 0$ 和 $w(e_0) = 1$。此外，我们在数据流开头插入一个 $+\infty$ 元素，该元素被视为大于任何其他元素，并以 e_s 的形式存储在 WQS 中。该元素的 r-最小值和 r-最大值也始终等于已插入元素的权重之和（包括其自身）。由于 $+\infty$ 是最大的元素，将其插入 S_w 不会影响任何其他元素的秩。

观察5.1. (r-min(e) + $j - 1$) 和 (r-max(e) + $j - 1$) 是 $e^{(j)}$ 秩的下界和上界 .

在数据流处理过程中，我们会向摘要中插入和删除元素。这会改变元素的秩，因此我们必须更新 WQS 以反映这些变化。下面介绍用于更新元素 r-min 和 r-max 值的过程:

Insert($x, w(x)$)。将给定元素 x （权重为 $w(x)$）插入 WQS。

(i) 将元素 x 及其权重 $w(x)$ 存储在 WQS 中。

(ii) 找出 WQS 中满足 $e_i > x$ 的最小元素 e_i；

(iii) 将 r-min(x) =r-min(e_{i-1}) + $w(e_{i-1})$，并将 r-max(x) =r-max(e_i)；此外，对所有 $j \geqslant i$，将 r-min(e_j) 和 r-max(e_j) 增加 $w(x)$。

删除(e_i)。从 WQS 中删除元素 e_i。

(i) 从分位数摘要中移除元素 e_i；保持所有其余 r-min、r-max 值不变。

现在，我们说明执行上述操作后，摘要中的每个元素 e 的 r-min 和 r-max 值都是元素 $e^{(1)}$ 的秩的有效下界和上界。假设将一个满足 $e_{i-1} < x < e_i$ 的新元素x 插入 WQS。x 的秩至少比 e_{i-1}的最后一个副本的秩大一。因此，r-min(x) 被设为 (r-min(e_{i-1}) + $w(e_{i-1})$ − 1) + 1 = r-min(e_{i-1}) + $w(e_{i-1})$，这是 $e^{(1)}$的秩的有效下界。x 的秩至多等于e_i的第一个副本的秩。因此，将 r-max(x) 设为等于 r-max(e_i)，即可使其成为有效上界。插入 x后，摘要中所有大于 x 的元素的秩都会增加 $w(x)$，因此需要更新它们的 r-min 和 r-max 值。小于 x 的元素的秩则不会改变。此外，从摘要中删除一个元素不会改变摘要中其他元素的秩界。

以下命题表明，如果 WQS 中元素的 r-min 和 r-max 值满足某个条件，就能保证 WQS 是 Unfold(S_w) 的 ε-近似摘要。

命题 5.2. 假设在长度为 n 的数据流上构建的 WQS 满足 r-最大值(e_i) − (r-min(e_{i-1}) + $w(e_{i-1})$ − 1) $\leqslant \lfloor \varepsilon W_n \rfloor$; 那么 WQS 是 Unfold($S_w$) 的 ε-近似分位数摘要。

证明。首先注意，对于每个 e，WQS 都存储了足够的信息，可以重建 Unfold(S_w) 中所有 e 副本的 r-min 和 r-max 值。因此，在本证明中，我们假设 e 的所有副本实际上都存在于 WQS 中。

根据命题 2.1，只需证明：对于任意两个相邻的元素 e 和 e' （属于）WQS 且满足$e \geqslant e'$，有 r-最大值(e) −r- 最小值 (e') $\leqslant \lfloor \varepsilon W_n \rfloor$。若 e 和 e' 相等，即 $e = e_i^{(j)}$ 和 $e' = e_i^{(j-1)}$ 对于某个

$i \in [s]$ 和 $2 \leqslant j \leqslant w(e_i)$，则有：

$$\begin{aligned}\text{r-max}(e) - \text{r-min}(e') &= (\text{r-max}(e_i) + j - 1) - (\text{r-min}(e_i) + j - 2) \\ &= \text{r-max}(e_i) - (\text{r-min}(e_i) - 1) \\ &\leqslant \text{r-max}(e_i) - (\text{r-min}(e_{i-1}) + w(e_{i-1}) - 1) \\ &\qquad\qquad (\text{From r-min}(e_i) \geqslant \text{r-min}(e_{i-1}) + w(e_{i-1})) \\ &\leqslant \lfloor \varepsilon W_n \rfloor.\end{aligned}$$

剩下的情况是，当 $e = e_i^{(1)}$ and $e' = e^{(w(e_{i-1}))\ i-1}$ 对某个 $i \in [s]$成立时。

$$\text{r-max}(e) - \text{r-min}(e') = \text{r-max}(e_i) - (\text{r-min}(e_{i-1}) + w(e_{i-1}) - 1) \leqslant \lfloor \varepsilon W_n \rfloor. \quad \blacksquare$$

5.1 时间步和带

我们为加权数据流定义一个等价的时间步概念。我们称，在 $t_k = \lfloor \varepsilon W_k \rfloor$ 个元素到达后，已经过去了 k 个时间步， S_w。直观而言，总权重为 $\ell := \frac{1}{\varepsilon}$ 的数据块在一个时间步内到达数据流。对于每个元素 x_k，在 S_w 中，我们定义其插入时间步 $t_0(x_k) = \lfloor \varepsilon(W_{k-1} + 1) \rfloor$。元素 x 在 S_w 中的带值，是根据 x 在 Unfold(S_w) 中的第一个副本，依据定义 2.3所分配的带值。

对于加权设置，我们还将给出带的一个等价形式化定义，以便在 O（1）时间内计算这些带。关于两者等价性的证明，请参阅引理 2.6 （见第 3 节）。

定义 5.3（带值与带）。插入了 k 个 S_w 后，对于数据流中的任意元素 x，当且仅当满足以下不等式时，b 值(x) 为 α，

$$2^{\alpha-1} + (t_k \bmod 2^{\alpha-1}) \leqslant t_k - t_0(x) < 2^{\alpha} + (t_k \bmod 2^{\alpha}).$$

对于任意整数 $\alpha \geqslant 0$，我们将所有 b 值(x) 为 $= \alpha$ 的元素 x 所构成的集合称为带 α，记作 Band$_\alpha$；我们也用 Band$_{\leqslant \alpha}$ 表示从带 0 到 α 的并集。

定义 5.3 的一个推论是，带值的数量在观察到 k 个元素后为 $B^{(k)} = O(\log t_k) = O(\log(\varepsilon W_k))$。我们还注意到，在任意时刻，属于带 0 到 α 的所有元素的权重之和至多为 $O(\ell \cdot 2^{\alpha+1})$，因为这些元素的所有副本都属于 Band$_{\leqslant \alpha}$，对于展开$(S_w$。我们将这些事实列于下文：

$$\#\text{ of b-values } B^{(k)} = O(\log \varepsilon W_k) \quad \text{and} \quad \sum_{x \in \text{Band}_{\leqslant \alpha}} w(x) \leqslant O(\ell \cdot 2^{\alpha+1}) \text{ for all } \alpha \geqslant 0. \tag{13}$$

现在我们给出以下观察：

观察 5.4。在任意时刻，如果元素 x 和 y 的 b 值(x) 6 b 值(y)，那么此后任意时刻，b 值(x) 6 b 值(y)。

这仅仅是因为，在无权设置中，元素的带值会同时根据当前时间步的值进行更新（定义 2.3）。因此，在带中的每个元素的第一个副本的 b 值也会在 展开(S_w) 中同时更新。因此，观察 5.4 成立。

5.2 r-min 和 r-max 值的间接处理

为了描述我们的算法，最好间接存储 r-min 和 r-max 值，将其表示为 g 和 Δ 值，并将它们在加权算法中的定义如下。对于 WQS 中的任意元素 e_i，

$$g_i = \text{r-min}(e_i) - (\text{r-min}(e_{i-1}) + w(e_{i-1}) - 1), \qquad \Delta_i = \text{r-max}(e_i) - \text{r-min}(e_i); \tag{14}$$

g 值可以解释为 e_i 的可能的最小秩与 e_{i-1}的最后一个副本的可能的最小秩之差。 Δ 值是 r-最大值与 r-最小值之差

e_i第一个副本的值。给定 WQS 中所有元素的 g值、 Δ值和权重，可以按如下方式恢复 r-min 和 r-max 值：

$$\text{r-min}(e_i) = g_i + \sum_{j=1}^{i-1}(g_j + w(e_j) - 1), \qquad \text{r-max}(e_i) = \Delta_i + g_i + \sum_{j=1}^{i-1}(g_j + w(e_j) - 1).$$

这促使我们为 WQS 中的每个元素 e_i 定义量 G_i：

$$G_i = g_i + w(e_i) - 1. \tag{15}$$

我们很快就会看到， G值具有一个很好的性质，这一性质在分析我们提出的算法时将很有用。现在，我们用已定义的 g 值和 Δ 值来陈述我们要维持的不变量，以确保 WQS 是一个 ε近似分位数摘要。

不变量 2. 在看到 S_w中的 k 个元素后，WQS 中的每个元素 $e_i \in$ 都满足 $g_i + \Delta_i \leqslant t_k$。

根据 公式（14），我们注意到 $g_i + \Delta_i =$ r-最大值(e_i) –(r-最小值(e_{i-1}) $+ w(e_{i-1}) - 1$。此外，$t_k = \lfloor \varepsilon W_k \rfloor$ 根据定义。因此，若 WQS 维持不变量 2，命题 5.2意味着它是一个 ε近似分位数摘要，满足 S_w。

下面的观察说明了在 g 和 Δ 操作期间，元素的值如何更新插入和删除（参见第 5 节）。

观察 5.5。在摘要 WQS 中：

- 插入$(x, w(x))$：设置 $g(x) = 1$和 $\Delta(x) = g_i + \Delta_i - 1$，并保持其余 (g, Δ) 值不变。
- 删除(e_i)：将 g_{i+1} 设置为等于 $g_{i+1} + G_i = g_{i+1} + (g_i + w(e_i) - 1)$，并保持其余 (g, Δ) 值不变。

观察 5.5 的正确性源于公式（14）以及执行这些操作时 r-min 和 r-max 值的变化方式。正如所承诺的，我们给出 G 和 Δ 值的一些有用性质。

***G*-值。**为了理解这一点，我们定义 WQS 中任意元素的覆盖概念。我们称 e_i 覆盖e_{i-1}，当且仅当 e_{i-1} 从摘要中删除；此时， e_i 也会覆盖 e_{i-1} 此前所覆盖的所有元素（每个元素在插入时只覆盖自身）。我们定义

- $C(e_i)$：由 e_i 覆盖的元素集合。根据定义，在任何时刻

$$G_i = \sum_{x \in C(e_i)} w(x) \text{ and } C(e_i) \cap C(e_j) = \emptyset \tag{16}$$

对于当前存储在分位数摘要中的任意 e_i、 e_j。

我们声称， G_i 等于 e_i覆盖范围内所有元素的权重之和。这很容易通过归纳法验证。当我们插入一个元素时，会将它的 g 值设为 1，且该元素只覆盖自身，因此根据 方程 (15)，它的 G 值等于其权重。在按照观察 5.5更新删除过程中的 G值时，这一性质在整个算法过程中始终成立。

Δ值。元素的 Δ值是衡量我们对其秩的了解误差的指标。我们可以利用不变量 2 推导出元素 Δ 值的以下上界， x 其插入时间为 $t_0(x)$。

$$\Delta(x) \leqslant t_0(x). \tag{17}$$

这里的直觉是，在观察到数据流中的 k 个元素后，如果保持不变量 2，那么可能秩之间的最大差异将受 t_k 限制。因此，新插入元素的秩误差也受 t_k 限制。形式化地说，假设 x 是数据流中的第 j 个元素，并且在插入 WQS 时满足 $e_{i-1} < x < e_i$。将 x 插入 WQS 时，我们设置 $\Delta(x) = g_i + \Delta_i - 1$。不变量 2 意味着 $\Delta(x) \leqslant \lfloor \varepsilon W_{j-1} \rfloor - 1 \leqslant \lfloor \varepsilon (W_{j-1} + 1) \rfloor = t_0(x)$。

6 加权数据流的 GK 算法的非平凡扩展

本节中，我们介绍 GK 算法针对加权数据流的扩展。回顾第 3.1 节 中的讨论，我们阐明了 GK 摘要中一些违反直觉的选择，而这些选择最终成为其更紧致的 $O(\frac{1}{\varepsilon}\log(\varepsilon n))$ 空间的基础。具体而言，我们给出以下定义：

定义 6.1（段）。WQS 中元素 e_i 的段记为 seg(e_i)，定义为 WQS 中一组最大的连续元素 e_j、e_{j+1}、$\cdots$、e_{i-1}，其 b 值严格小于 b 值(e_i)。令 G_i^* 为 e_i 及其所在段中 G 值的总和，即 $G_i^* = G_i + \sum_{e_k \in \text{seg}(e_i)} G_k$。

见 图 5 下方的示意图。

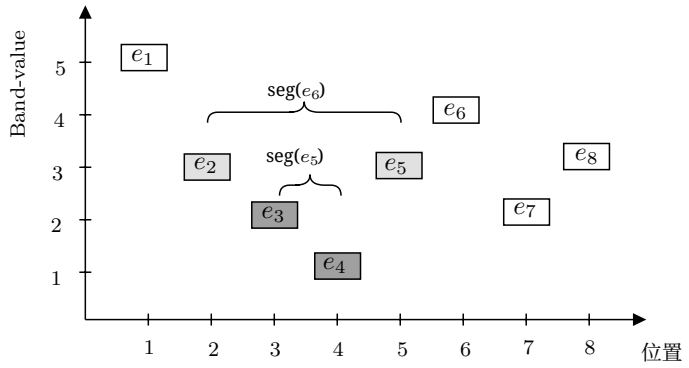


图 5：定义定义 6.1中，元素的秩沿水平方向递增。元素 e_5 的段包含 e_3 和 e_4。元素 e_6 的段包含 e_2、e_3、e_4 和 e_5。

每一步中，算法首先将到达的元素插入 WQS；我们称之为插入步骤。随后，只有当可以从 WQS 中删除某个元素，且同时删除其整个段而不违反不变量 2时，算法才会删除该元素。只要存在这样的元素，删除该元素（及其所在段）不会违反不变量 2（以及另一项关于 b 值的简单但重要的条件），算法就会将其从 WQS中删除；我们称之为删除步骤。现在我们正式描述该算法。

算法 3. 用于加权数据流的 GK 算法推广：

对于每个到达的项（x_j、w（x_j））：

（i）运行 Insert（x_j、w（x_j））：

（ii）当 QS 中存在满足以下条件的元素 e_i 时：

$$(1)\ \text{b-value}(e_i) \leqslant \text{b-value}(e_{i+1}) \quad \underline{\text{and}} \quad (2)\ G_i^* + g_{i+1} + \Delta_{i+1} \leqslant t_j$$

对 $\{e_i\} \cup \text{seg}(e_i)$ 中的 e_k 执行删除(e_k)。

定理 3。对于任意 $\varepsilon > 0$ 和总权重为 W_n、长度为 n 的加权数据流，算法 3 可在 $O(\frac{1}{\varepsilon} \cdot \log(\varepsilon W_n))$ 空间内维护一个 ε-近似分位数摘要。此外，存在一种算法 3 的实现，每个元素的最坏情况更新时间为 $O\big(\log(1/\varepsilon) + \log\log(\varepsilon W_n) + \log^2(\frac{\varepsilon W_n}{\varepsilon n})\big)$。

我们在此指出，只要权重以多项式 n 为界，且 $\varepsilon \geqslant 1/n^{1-\delta}$ 对任意固定的 $\delta \in (0,1)$，算法所用的空间为 $O((1/\varepsilon)\log(\varepsilon n))$，更新时间为 $O(\log(1/\varepsilon)+\log\log(\varepsilon n))$。这与 [LWYC16] 中描述的 GK 摘要实现的空间和时间复杂度相符。请注意，ε 有意义的取值范围是至少为一个较小的常数，因为当 $\varepsilon < 1/n^{1-\delta}$ 时，摘要大小的 $(1/2\varepsilon)$ 信息论下界已意味着，即使对无权输入的原始 GK 摘要，也需要存储 $\Omega(n^{1-\delta})$ 个元素，这对大多数应用来说是难以承受的。

算法 3 维护了一个有效的 ε-近似摘要，因为它只有在条件 (e_i): ii 成立时，才会连同其所在段一起删除一个元素 $G_i^* + g_{i+1} + \Delta_{i+1} \leqslant t_k$。因此，删除 不变量 2 对元素 e_{i+1} 后，该不变量仍然成立；删除 e_i (其他 g 和 Δ 值不受影响）。接下来，我们重点分析算法所用空间的上界。然后，在 第 6.2 节中，我们将给出一种高效实现，以完成定理 3的证明。

6.1 空间分析

在本小节中，我们将证明 算法 3所使用空间的上界。具体来说，我们有以下结论：

引理 6.2。 对于任意 $\varepsilon > 0$ 以及长度为 n 且总权重为 W_n 的数据流，算法 3 维护一个ε-近似分位数摘要，其所需 $O(\frac{1}{\varepsilon} \cdot \log(\varepsilon W_n))$ 空间。

我们首先提出一个关键观察。

观察 6.3。WQS 中来自带$\leqslant\alpha$ 的元素在任意时刻只覆盖带$\leqslant\alpha$ 中的元素。

这是因为当 e_i 和 seg(e_i) 被 e_{i+1} 覆盖时，算法 3 确保 b 值(e_i) $\leqslant$ b 值(e_{i+1})。根据定义 6.1，seg(e_i)包含 b 值小于 b 值(e_i)的元素。因此， $C(e_{i+1})$包含 b 值至多为 b 值(e_{i+1})的元素；根据观察 5.4，此性质在之后的时刻仍然成立。

另一个重要的观察是，在执行删除步骤之后，经过 k 次插入，元素 e_i若存在于 WQS 中，则满足 b 值(e_i) $>$ b 值(e_{i+1}) 或 $G_i^* + g_{i+1} + \Delta_{i+1} > t_k$；否则 算法 3就会删除该元素。我们将 WQS 中满足前一个条件的元素称为 1 型元素，将仅满足后一个条件的元素称为 2 型 元素。因此，每个元素恰好属于这两种类型之一（只有 $e_s = +\infty$ 除外，我们可以忽略它）。因此，只要界定 1 型和 2 型元素的数量，就足以界定 WQS的空间复杂度。我们先通过以下引理界定 1 型元素的数量。

引理 6.4. 在看到 k 个元素后执行删除步骤时，WQS 中存储的 1 型元素数量为 $O(\ell \cdot \log t_k)$。

证明。我们首先将类型 1 元素划分为 $B^{(k)}$ 个集合 Y_0, ..., $Y_{B^{(k)}}$，其中对于任意带值 α:

$$Y_\alpha := \{e_i \in \texttt{WQS} \mid e_i \text{ is type-1 and b-value}(e_{i+1}) = \alpha\};$$

(注意， Y_α 中的元素满足其后继元素的带值为 α，而不是它们自身的带值[3]) 我们将证明，任意集合 Y_α 的大小至多为 $O(\ell)$。我们将 e_i 中的每个元素映射到 e_j 中 b 值大于 α的最小元素；示意图见 图 6a 。令 T_α 为所有这类元素 e_j构成的集合。此外，不难看出，从 Y_α 到 T_α 的映射是一一对应的；因此 $|Y_\alpha| = |T_\alpha|$。注意， e_{j-1} 必定是一个 2 型元素。因此，

$$G_{j-1}^* + g_j + \Delta_j > t_k. \tag{18}$$

由于 b 值(e_j) 大于 b 值(e_{i+1}) $= \alpha$，根据 观察 5.4，可以推知 e_j 在 WQS中的插入时间早于 e_{i+1}。令 g'_j 表示 e_j 插入时的 g 值。根据 不变量 2，

$$g'_j + \Delta_j \leqslant t_0(e_{i+1}) \quad (\Delta \text{ value does not change over time}). \tag{19}$$

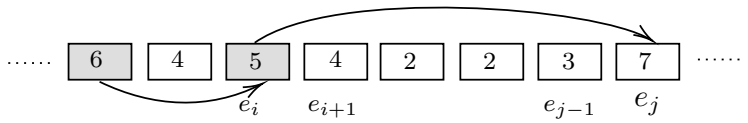
将 等式 (19) 从 等式 (18)中相减，并利用 定义 5.3 中的界，我们得出

$$G_{j-1}^* + (g_j - g'_j) > t_k - t_0(e_{i+1}) \geqslant 2^{\alpha-1} - 2. \tag{20}$$

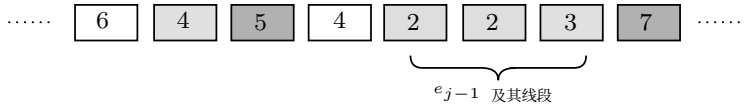
在上面的方程中：

(i) 术语 $(g_j - g'_j)$ 表示 e_j 插入 e_{i+1} 后， e_j 所覆盖元素的权重总和。断言 6.5 将证明这些元素都位于带$\leqslant\alpha$中。

[3]乍看之下，这可能听起来有些违背直觉，但请记住，元素类型的判定标准取决于该元素及其下一个元素；这个定义使我们能够将这一点考虑在内。



(a) 阴影方块是 Y_4 的元素。箭头表示从 Y_4 中的元素到 T_4 中元素的映射。 Y_4 中的每个元素 e_i 都映射到第一个比它大的、带值高于 4 的元素 e_j 。



(b) 每个深灰色块表示 e_j 中的一个元素。所有满足以下任一条件的元素：e_{j-1} 或位于 seg(e_{j-1}) 中，均以浅灰色阴影标出。

图6：这两幅图表示摘要的一部分，其中每个方块代表一个元素。方块中的数字是该元素的带值。

(*ii*) 术语 G^*_{j-1} 表示被 e_{j-1} 和 seg(e_{j-1}) 覆盖的元素权重之和。根据定义 6.1， e_{j-1} 以及 seg(e_{j-1}) 中的所有元素的 b 值均为 $\leqslant \alpha$。 观察 6.3 使我们能够得出结论：由 G^*_{j-1} 计入的元素权重之和也属于带$\leqslant\alpha$ 。

(*iii*) 此外，很容易看出，对于 e_{j_1} 和 e_{j_2} 中彼此不同的元素 T_α， e_{j_1-1} 和 e_{j_2-1} 的线段不会重叠（如图6所示）。因此，根据公式（16），被 e_{j_1} 及其线段覆盖的元素，与被和 e_{j_2} 及其线段覆盖的元素互不相同。

最后，根据上述讨论，我们得出，公式（20）的左端对所有 T_α求和后，与带$\leqslant\alpha$中所有元素的总权重成正比。形式化地说，

$$|T_\alpha| \cdot (2^{\alpha-1}-2) \leqslant \sum_{e_j \in T_\alpha} G^*_{j-1} + \sum_{e_j \in T_\alpha} (g_j - g'_j) \leqslant \sum_{x_k \in \mathrm{Band}_{\leqslant\alpha}} w(x_k) + \sum_{x_k \in \mathrm{Band}_{\leqslant\alpha}} w(x_k) \leqslant O(\ell \cdot 2^{\alpha+2}),$$

最后一个不等式由公式 (13)得出。因此，对于 $|T_\alpha| = |Y_\alpha| = O(\ell)$，有 $\alpha \geqslant 3$。对于 $O(\ell)$，我们有 $\alpha = 0$、1、2 个元素。由于 $B^{(k)} = \log t_k$是 α的可能取值数量，1 型元素的数量为$O(\ell \log t_k)$。

断言 6.5. A　在插入 e_{i+1} 之后， e_j 覆盖的所有元素当前的 b 值至多为 α 。

证明。假设存在一个元素，其当前的 b 值为 $> \alpha$ ，但在插入 e_{i+1} 之后被 e_j覆盖。由于这些元素会被 e_j覆盖，因此它们都小于 e_j 且大于 e_{i+1} 。现在考虑其中最小的元素 e。显然，在插入 e_{i+1}之后， e_{i+1} 属于 e 的段。由于 e 目前不属于摘要，因此它一定已被删除。这意味着包含 e_{i+1}的段已被删除。这说明 e_{i+1} 也已被删除，矛盾。

∎ 断言 6.5

这就完成了引理 6.4的证明。 ∎

现在还需要　界定 WQS 中 2 型元素的数量，我们将在下面的引理中完成这项工作。

引理 6.6. 在数据流中已看到 k 个元素后，执行删除步骤，此时 2 型元素的数量为 $O(\ell \cdot \log t_k)$ 。

证明。WQS 中的任意 2 型元素 e_i 都具有性质 $G^*_i + g_{i+1} + \Delta_i > t_k$。这将给出一个关于 b 值（$e_{i+1}$）的下界 $G^*_i + g_{i+1}$ 。

命题 6.7. 观察到 k 个元素后，对于任意 2 型元素 e_i, $G^*_i + g_{i+1} \geqslant 2^{\text{b-value}(e_{i+1})-1} - 2$。

证明。由于 e_i 是一个 2 型元素，$G_i^* + g_{i+1} + \Delta_{i+1} > t_k$。根据公式（17），$\Delta_{i+1} \leqslant t_0(e_{i+1})$，因此，

$$G_i^* + g_{i+1} > t_k - t_0(e_{i+1}) \geqslant 2^{\text{b-value}(e_{i+1})-1} - 2,$$

其中第二个不等式根据定义 5.3。 ∎ 命题 6.7

命题 6.7 给出了每个 2 型元素 G^*的下界，该下界是 e_i 的 g值和带值的函数，其中 下一个元素 e_{i+1}。因此，我们将类型 2 元素划分为集合 X_0、……、$X_{B^{(k)}}$，使得对于任意带值 α，

$$X_\alpha := \{e_i \in \texttt{WQS} \mid e_i \text{ is type-2 and b-value}(e_{i+1}) = \alpha\}.$$

此外，对于任意 $e_i \in X_\alpha$，由于 e_i 是一个类型 2 元素，b 值(e_i) $\leqslant$ b 值(e_{i+1}) $= \alpha$。对 命题 6.7 在 X_α中的每个元素对应的不等式求和，可得：

$$|X_\alpha| \cdot (2^{\alpha-1} - 2) \leqslant \sum_{e_i \in X_\alpha} G_i^* + g_{i+1}. \tag{21}$$

接下来，我们给出 方程 (21) 右侧的一个上界，由此可推出$|X_\alpha|$所需的界。

命题 6.8。 在看到 k 个元素后，对于任意 $\alpha \geqslant 0$，$\sum_{e_i \in X_\alpha} G_i^* \leqslant 2 \sum_{e_j \in \texttt{WQS} \cap \text{带} \leqslant \alpha} G_j$.

证明。我们将 X_α划分为两个不相交的集合 $X_\alpha \cap$带$_\alpha$和 $X_\alpha \cap$带$_{\leqslant \alpha-1}$，并注意到，这两个集合中的某个集合必定包含两个线段不相交的元素。此外，它们所在的线段中的元素必定都属于带$_{\leqslant \alpha}$。因此，

$$\begin{aligned}\sum_{e_i \in X_\alpha} G_i^* &= \sum_{e_i \in X_\alpha \cap \text{Band}_\alpha} G_i^* + \sum_{e_i \in X_\alpha \cap \text{Band}_{\leqslant \alpha-1}} G_i^* \leqslant \sum_{e_j \in \texttt{WQS} \cap \text{Band}_{\leqslant \alpha}} G_j + \sum_{e_j \in \texttt{WQS} \cap \text{Band}_{\leqslant \alpha}} G_j. \\ &= 2 \sum_{e_j \in \texttt{WQS} \cap \text{Band}_{\leqslant \alpha}} G_j. \end{aligned}$$ ∎ Claim 6.8

下一个命题对 WQS 中来自带$_{\leqslant \alpha}$的元素的 G 值之和给出界。

命题 6.9。看到 k 个元素后，对于任意 $\alpha \geqslant 0$, $\sum_{e_i \in \texttt{WQS} \cap \text{带}_{\leqslant \alpha}} G_i \leqslant O(\ell \cdot 2^{\alpha+1})$。

证明。 只有在满足条件 (1) 时，算法 3 才会删除元素。根据观察 5.4，在算法后续的任何时刻，这一条件仍然成立。因此，$C(e_i)$ 仅包含 b 值至多为 b 值(e_i) 的元素。因此，

$$\begin{aligned}\sum_{e_i \in \texttt{WQS} \cap \text{Band}_{\leqslant \alpha}} G_i &= \sum_{e_i \in \texttt{WQS} \cap \text{Band}_{\leqslant \alpha}} \sum_{x_j \in C(e_i)} w(x_j) && (\text{as } G_i = \textstyle\sum_{x_j \in C(e_i)} w(x_j) \text{ by Eq (16)}) \\ &\leqslant \sum_{x_j \in \text{Band} \leqslant \alpha} w(x_j) && (\text{as } C(e_i)\text{'s are disjoint and their elements belong to Band}\leqslant \alpha) \\ &= O(\ell \cdot 2^{\alpha+1}), && (\text{by the bound in Eq (13)})\end{aligned}$$

完成论证。 ∎ 命题 6.9 ∎

将 命题 6.8 和 命题 6.9 的界代入 方程 (21)，并利用一个 G 元素的值至少为其 g 值这一事实，可得：

$$|X_\alpha| \cdot (2^{\alpha-1} - 2) \leqslant \sum_{e_i \in X_\alpha} G_i^* + \sum_{e_j \in \texttt{WQS} \cap \text{Band}_{\leqslant \alpha}} g_j \leqslant 2 \sum_{e_j \in \texttt{WQS} \cap \text{Band}_{\leqslant \alpha}} G_j + \sum_{e_j \in \texttt{WQS} \cap \text{Band}_{\leqslant \alpha}} G_j \leqslant 3 \cdot O(\ell \cdot 2^{\alpha+1}),$$

这意味着 $|X_\alpha| = O(\ell)$ 对于 $3 \leqslant \alpha \leqslant B^{(k)}$。在 X_0 中，每个都至多有 $O(\ell)$ 个元素，X_1 并且 X_2 由于带中至多有 $O(\ell)$ 个元素$\leqslant 2$。根据 公式 (13) 可得 $B^{(k)} = O(\log t_k)$，因此 2 型元素的数量为 $O(\ell \cdot \log t_k)$。 ∎

现在我们已经证明，在观察到 k 个元素后执行删除步骤时，WQS 中的类型 1 元素数量为 $O(\ell \cdot \log\ t_k)$，依据是 引理 6.4；根据 引理 6.6，WQS 中的 2 型元素数量为 $O(\ell \cdot \log t_k)$。由于 WQS中的每个元素（除 $+\infty$外）要么是类型 1，要么是类型 2，因此 WQS 中的元素总数为 $O(\ell \cdot \log t_k)$。

至此，引理 6.2 的证明 $t_n = O(\varepsilon W_n)$ 和 $\ell = O(\frac{1}{\varepsilon}$完成。我们以以下备注结束对空间复杂度的讨论；

备注 6.10 (延迟删除)。 假设在算法 3中，不再于每个元素插入后运行删除步骤，而是仅在插入 c 个元素 $c > 1$后才运行第 (ii) 行中的删除步骤；那么，该算法的空间复杂度仅增加一个加性项 $O(c)$。

在 k 个元素后执行删除步骤，元素数量会减少到 $O(\ell \cdot \log t_k)$，正如前文所证明的，只要我们在每次删除时都满足算法 3 中关于删除的两个条件。因此，额外的空间仅用于存储插入到 $O(c$中的额外WQS 个元素。

上述备注有助于提出一种算法 3 的实现，该实现能使每个元素的更新时间渐近地更快，下面将对此进行说明。

6.2 算法 3的高效实现

本节将介绍 算法 3的一种高效实现。它与 [LWYC16] 中提出的 GK 摘要实现类似。关键思路在于，删除步骤比较耗时，因此每个时间步之后都执行删除操作会导致时间效率较低。然而，若长时间不执行删除步骤，空间占用就会急剧增加。我们提出的快速实现处理了这一权衡，并选择连续删除步骤之间的延迟，使算法的时间和空间复杂度都达到最优。形式化地，我们证明如下结论：

引理 6.11。 存在一种 算法 3 实现，其每个元素的最坏情况处理时间为 $O\left(\log(1/\varepsilon) + \log\log(\varepsilon W_n) + \log^2(\frac{\varepsilon W_n}{\varepsilon n})\right)$。

第一部分：存储分位数摘要：我们将 WQS 摘要存储为一棵平衡二叉搜索树（二叉搜索树），其中每个节点都包含 WQS 的一个元素及其元数据。对于每个元素 e，我们存储 $w(e)$ $g(e)$ $\Delta(e)$ 和 $t_0(e)$。二叉搜索树的排序键是元素的值。插入和删除操作分别将元素插入二叉搜索树和从中删除元素。

第二部分：执行删除步骤：删除步骤涉及删除摘要中满足以下两个条件的元素：算法 3。检查条件 (ii) 要求我们知道与摘要中每个元素对应的 G^*值。我们首先说明如何计算所有元素的 G^* 值。

计算 G^* 值：首先，我们对 WQS 进行中序遍历，并将元素 e_i 按排序顺序存储为一个临时链表。计算 G^* 值时会用到一个栈，并从最小元素到最大元素对链表进行一次遍历。下面描述遍历到链表中的元素 e_i 时的计算过程。为了得到 G_i^*，我们将栈顶所有 b 值小于 b 值(e_i)的元素的 G^* 值相加，再将所得总和加到 G_i 上。随后，将这些元素全部从栈中弹出，并将 e_i 及其计算所得的 G^* 值压入栈中。我们声称，此时 G_i^* 已被正确计算。由于每个元素至多入栈和出栈一次，因此所有元素的 G^* 值都可以在与 WQS 大小呈线性关系的时间内计算出来。

现在我们说明如何执行每个删除步骤。

算法。高效执行删除步骤：

1. 对WQS （它是一棵二叉搜索树）进行中序遍历，得到一个按值排序的临时（双向链表）元素列表。2. 使用WQS 和定义 5.3计算所有元素的 G^* 值。3. 使用上述算法计算所有元素的 b 值。4. 从较大的元素向较小的元素遍历列表。对于每个元素 e_i，如果它同时满足算法 3中提到的两个删除条件，则将其从二叉搜索树（以及列表）中删除。

下面给出一种算法 3 的实现，具有较快的摊销更新时间。我们还将介绍如何修改这一实现，使其最坏情况更新时间也达到相同的界。

实现 3：算法 3 的高效实现算法 3

- 将 WQS 初始化为空的平衡二叉搜索树。
- 删除时间 $\leftarrow 2$。
- 对于每个到达的项（$x_k, w(x_k)$）：(*i*) 执行插入($x_k, w(x_k)$)。(*ii*) 如果（k =删除时间）：– 执行删除步骤并更新删除时间 $\leftarrow$ 删除时间 $+ \lceil \ell \log\ t_k \rceil$。

空间分析。 上述实现的空间复杂度仍为 $O(\frac{1}{\varepsilon}\log(\varepsilon W_n))$。这是因为，当我们看到 k 个元素后执行删除操作时，只需再等待 $O(\ell \cdot \log t_k)$ 个元素；根据 备注 6.10，这只会使空间复杂度增加一个常数因子。因此，经过 n 次插入后，空间复杂度仍为 $O(\ell \cdot \log\ t_n) = O(\frac{1}{\varepsilon}\log(\varepsilon W_n))$，因为 $\ell = 1/\varepsilon$ 且$t_n = O(\varepsilon W_n)$。

时间分析。将 WQS 存储为二叉搜索树的主要目的是减少在 WQS 上执行插入和删除操作所需的时间。该操作只需 $O(\log s)$，其中 s 是摘要大小，至多为 $O(\frac{1}{\varepsilon}\log\ \varepsilon W_n)$。因此，我们现在可以得出以下结论：由每个元素至多执行一次插入和删除操作这一事实可直接推出这些结论。

观察 6.12. 对于长度为 n的数据流，实现 3的快速实现执行所有插入和删除操作所花费的总时间为 $O(n \cdot (\log(1/\varepsilon) + \log\log(\varepsilon W_n)))$。

请注意，实现 3 未计入观察 6.12 的唯一耗时部分，是用于确定要删除哪些元素的部分；我们将在下文对此进行界定。

引理 6.13. 对于长度为 n的数据流，实现 3 在所有已执行的删除步骤中，确定哪些元素需要删除所花费的总时间为 $O(n + \frac{1}{\varepsilon}\log^2(\varepsilon W_n))$。

证明。一次删除步骤中（已见到 k 个元素时）确定哪些元素需要删除所花费的时间为 $O(s) = O(\ell \cdot \log t_k)$。这是因为，创建链表，随后计算所有元素的 b 值和 G^* 值，可以在 $O(s)$ 时间内完成。最后，从最大元素到最小元素对列表进行线性遍历（以检查删除条件是否成立）需要 $O(s)$ 时间。

接下来，我们推导算法执行的删除步骤数量的上界。考虑当 t_k 为区间 $[2^i, 2^{i+1})$ 时执行的删除步骤，其中 $1 \leqslant i \leqslant \lceil \log(\varepsilon W_n) \rceil$。令 $d(i)$ 表示此类删除步骤的数量，令 $n(i)$ 表示数据流中满足 t_k 位于范围内的元素 x_k 的数量

$[2^i, 2^{i+1})$。在看到 k 个元素后执行删除步骤，我们会等待 $\lceil \ell \log\, t_k \rceil$ 次插入。因此，在该区间内发生的任意两个连续删除步骤之间，至少插入了 $\ell \cdot i$ 个元素。因此，我们得到该区间内执行的删除步骤数量的如下上界。

$$d(i) \leqslant \frac{n(i)}{\ell \cdot i} + 1. \tag{22}$$

在删除步骤中（看到 k 个元素后）决定删除哪个元素所花费的时间至多为$O(\ell \log\, t_k) = O(\ell \cdot i)$，其中 t_k 位于区间 $[2^i, 2^{i+1}$内。结合方程 (22)，可得出在所有删除步骤中，决定删除哪些元素所花费总时间的以下界限。

$$\begin{aligned} O\left(\sum_{i=1}^{\lceil \log(\varepsilon W_n)\rceil} d(i) \cdot \ell i\right) &= O\left(\sum_{i=1}^{\lceil \log(\varepsilon W_n)\rceil} (n(i) + \ell i)\right) \\ &= O\left(n + \frac{1}{\varepsilon}\log^2(\varepsilon W_n)\right) \end{aligned}$$

至此，引理的证明完成。 ∎

观察 6.12 和命题 3.11 显然共同推出：实现 3处理长度为 n 的数据流所需的总时间为 $O\big(\, n \cdot (\log(1/\varepsilon) + \log\log(\varepsilon W_n)) + \frac{1}{\varepsilon}\log^2(\varepsilon W_n)\,\big)$。因此，每个元素的摊销更新时间为 $O\big(\log(1/\varepsilon) + \log\log(\varepsilon W_n) + \frac{\log^2(\varepsilon W_n}{\varepsilon n}\big)$。

我们可以运用将低效操作的耗时分摊到多个时间步这一标准思路，得到每个元素最坏情况更新时间的相同界限。具体做法是在执行下一次删除步骤之前，利用接下来的所有时间步处理当前删除步骤。形式化地说，有如下结论：

断言 6.14. 存在一种算法 3 的实现，其最坏情况更新时间为 $O\big(\log(1/\varepsilon) + \log\log(\varepsilon W_n) + \frac{\log^2(\varepsilon W_n}{\varepsilon n}\big)$。

致谢

我们感谢 Rajiv Gandhi 促成作者之间的合作，并在整个项目过程中给予支持。

参考文献

[ABED+21] Noga Alon、Omri Ben-Eliezer、Yuval Dagan、Shay Moran、Moni Naor 和 Eylon Yogev。《在线分类中的大数对抗性定律与最优遗憾》。载于 第53届 ACM SIGACT 计算理论研讨会论文集，第447–455页，2021年。1[ACH+12] Pankaj K. Agarwal、Graham Cormode、Zengfeng Huang、Jeff M. Phillips、Zhewei Wei 和 Ke Yi。《可合并摘要》。载于 第31届 ACM SIGMOD-SIGACT-SIGART数据库系统原理研讨会论文集，PODS 2012，美国亚利桑那州斯科茨代尔，2012年5月20日至24日，2012年。1，2[AMS96] Noga Alon、Yossi Matias 和 Mario Szegedy。《近似频率矩的空间复杂度》。载于 第二十八届 ACM 计算理论年度研讨会论文集，美国宾夕法尼亚州费城，1996年5月22日至24日，第20–29页，1996年。1[BLRV13] Andrej Brodnik、Alejandro L´opez-Ortiz、Venkatesh Raman 和 Alfredo Viola（编）。 空间高效的数据结构、数据流与算法——纪念 J. Ian Munro 66岁生日论文集，第8066卷，收录于 计算机科学讲义。Springer，2013年。 3

[CJP08] Amit Chakrabarti、T. S. Jayram 和 Mihai Patrascu。《随机排序流中选择问题的紧下界》。载于第十九届 ACM-SIAM离散算法研讨会论文集（SODA 2008），美国加利福尼亚州旧金山，2008年1月20日至22日，第720–729页，2008年。2[CKMS06] Graham Cormode、Flip Korn、S. Muthukrishnan 和 Divesh Srivastava。《用于数据流偏置分位数的高效空间与时间确定性算法》。载于第二十五届 ACM SIGACT-SIGMOD-SIGART数据库系统原理研讨会论文集，2006年6月26日至28日，美国伊利诺伊州芝加哥，第263–272页，2006年。2[CM04] Graham Cormode 和 S. Muthukrishnan。《改进的数据流摘要：Count-min 草图及其应用》。载于 LATIN 2004：理论信息学，第六届拉丁美洲研讨会，阿根廷布宜诺斯艾利斯，2004年4月5日至8日，论文集，第29–38页，2004年。3[CV20] Graham Cormode 和 Pavel Vesel´y。《基于比较的分位数摘要的紧下界》。载于第39届 ACM SIGMOD-SIGACT-SIGAI数据库系统原理研讨会论文集（PODS 2020），美国俄勒冈州波特兰，2020年6月14日至19日，第81–93页，2020年。1、2、3[FO15] David Felber 和 Rafail Ostrovsky。《一种随机化在线分位数摘要，使用 $O(1/\epsilon \cdot \log(1/\epsilon))$ 个字》。载于《近似、随机化与组合优化：算法与技术》，APPROX/RANDOM 2015，美国新泽西州普林斯顿，2015年8月24日至26日，第775–785页，2015年。2[GHR+12] Anna C Gilbert、Brett Hemenway、Atri Rudra、Martin J Strauss 和 Mary Wootters。《恢复简单信号》。载于2012年信息论与应用研讨会，第382–391页。IEEE，2012年。1[GHS+12] Anna C Gilbert、Brett Hemenway、Martin J Strauss、David P Woodruff 和 Mary Wootters。《通过隐私实现可重复使用的低误差压缩采样方案》。载于2012年 IEEE统计信号处理研讨会（SSP），第536–539页。IEEE，2012年。1[GK01] Michael Greenwald 和 Sanjeev Khanna。《分位数摘要的空间高效在线计算》。载于2001年 ACM SIGMOD数据管理国际会议论文集，美国加利福尼亚州圣巴巴拉，2001年5月21日至24日，第58–66页，2001年。1、2、3、4、6、13[GM09] Sudipto Guha 和 Andrew McGregor。《流顺序与顺序统计量：随机顺序流中的分位数估计》。SIAM计算杂志，38(5):2044–2059，2009年。2[GZ03] Anupam Gupta 和 Francis Zane。《列表中的逆序数计数》。载于第十四届 ACM-SIAM离散算法研讨会论文集，2003年1月12日至14日，美国马里兰州巴尔的摩，第253–254页，2003年。2[HT10] Regant Y. S. Hung 和 Hing-Fung Ting。《在数据流中查找 ε-近似分位数的 $\Omega(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$ 空间下界》。载于《算法学前沿》，第四届国际研讨会（FAW 2010），中国武汉，2010年8月11日至13日，论文集，第89–100页，2010年。1、2、3[HW13] Moritz Hardt 和 David P Woodruff。《线性草图对自适应输入的鲁棒性如何？》。载于第四十五届 ACM 计算理论年度研讨会论文集，第121–130页，2013年。1[ILL+19] Nikita Ivkin、Edo Liberty、Kevin Lang、Zohar Karnin 和 Vladimir Braverman。《小空间、低更新时间的流式分位数算法》。arXiv 预印本 arXiv:1907.00236，2019年。3[KLL16] Zohar S. Karnin、Kevin J. Lang 和 Edo Liberty。《数据流中的最优分位数近似》。载于 IEEE 第57届计算机科学基础年度研讨会（FOCS 2016），2016年10月9日至11日，美国新泽西州新不伦瑞克凯悦酒店，第71–78页，2016年。1、3

[LWYC16] Ge Luo、Lu Wang、Ke Yi 和 Graham Cormode。《数据流上的分位数：实验比较、新分析与进一步改进》。VLDB J., 25(4):449–472, 2016. 1, 2, 8, 21, 25[MNS11] Ilya Mironov、Moni Naor 和 Gil Segev。《对抗环境中的草图化》。SIAM Journal on Computing, 40(6):1845–1870, 2011. 1[MP78] J. Ian Munro 和 Mike Paterson。《有限存储下的选择与排序》。载于第 19 届计算机科学基础年会，安娜堡，密歇根州，美国，1978 年 10 月 16–18 日，第 253–258 页，1978 年。1, 2[MRL98] Gurmeet Singh Manku、Sridhar Rajagopalan 和 Bruce G. Lindsay。《单遍扫描且使用有限内存的近似中位数及其他分位数》。载于 SIGMOD 1998：ACM SIGMOD数据管理国际会议论文集，1998 年 6 月 2–4 日，西雅图，华盛顿州，美国，第 426–435 页，1998 年。1, 2[MRL99] Gurmeet Singh Manku、Sridhar Rajagopalan 和 Bruce G. Lindsay。《用于大型数据集顺序统计量空间高效在线计算的随机采样技术》。载于 SIGMOD 1999：ACM SIGMOD数据管理国际会议论文集，1999 年 6 月 1–3 日，费城，宾夕法尼亚州，美国，第 251–262 页，1999 年。2[NY15] Moni Naor 和 Eylon Yogev。《对抗环境中的布隆过滤器》。载于年度密码学会议，第 565–584 页。Springer，2015 年。1[SBAS04] Nisheeth Shrivastava、Chiranjeeb Buragohain、Divyakant Agrawal 和 Subhash Suri。《中位数及更多：传感器网络的新型聚合技术》。载于第 2 届嵌入式网络传感器系统国际会议论文集（SenSys 2004），美国马里兰州巴尔的摩，2004 年 11 月 3–5 日，第 239–249 页，2004 年。3[wik] 次线性算法开放问题列表——问题 2：分位数。https://sublinear.info/2。1

适用于加权数据流的贪心算法

本节将介绍一种时间高效的算法，它会从摘要中贪心地删除元素`WQS`，同时保持不变量 2，并使元素的 b 值满足与算法 3相同的条件。其思路非常相似：我们先将每个到达的元素插入`WQS`，然后执行删除步骤以缩小摘要的规模。唯一的区别是，我们不会连同元素所属的段一起删除它；相反，即使无法删除整个段，我们也会贪心地删除该元素。形式化地说，

算法 4. 一种适用于加权数据流的贪心算法：

对于每个到达的项 $(x_k, w(x_k))$：

(i) 运行插入$(x_k, w(x_k))$。

(ii) 对满足以下条件的任意元素 e_i，反复运行删除(e_i)：该元素属于 WQS :

(1) b 值(e_i) 6 b 值(e_{i+1}) and (2) $G_i + g_{i+1} + \Delta_{i+1} \leqslant t_k$

我们在以下定理中给出算法 4 的空间和时间复杂度。

定理 4。对于任意 $\varepsilon > 0$，以及总权重为 W_n、长度为 n 的加权数据流，算法 4 可在 $O(\frac{1}{\varepsilon} \cdot \log^2(\varepsilon W_n))$ 空间内维护一个 ε-近似分位数摘要。此外，存在一种算法 4 的实现，其每个元素的更新时间为 $O\left(\log(1/\varepsilon) + \log\log(\varepsilon W_n) + \frac{\log^3(\varepsilon W_n}{\varepsilon n}\right)$。

这里我们指出，在 $\varepsilon \geqslant \frac{1}{\sqrt{n}}$ 的假设下，且当权重有 poly(n) 上界时，该算法的空间复杂度和更新时间分别为 $O((1/\varepsilon)\log^2(\varepsilon n))$ 和 $O(\log(1/\varepsilon)+\log\log(\varepsilon n))$。

算法 4 仅当 e_i 时才删除一个元素 $G_i + g_{i+1} + \Delta_{i+1} \leqslant t_k$。根据 观察 5.5，可以得出$g_{i+1} + \Delta_{i+1} \leqslant t_k$ 在删除后成立，从而确立 不变量 2。其他 g 和 Δ 值保持不变）。如 第 5.2 节所示，维持 不变量 2 足以使 WQS 成为有效的 ε-近似分位数摘要。接下来，我们将证明 算法 4 在定理 4中所述的空间复杂度保证。

A.1 空间分析

我们借用 第 6.1 节 中的 1 型和 2 型元素概念，并稍作修改。执行删除步骤后（此时已看到数据流中的 k 个元素），WQS 中的每个元素都满足 b 值(e_i) > b 值(e_{i+1}) 或 $G_i + g_{i+1} + \Delta_{i+1} > t_k$；否则 算法 4 就会将其删除。满足前一个条件的元素被归类为 1 型元素，其余元素（仅满足后一个条件的元素）则被归类为 2 型元素。和之前一样，我们将分别界定 1 型和 2 型元素的数量。

引理 A.1. 在已看到 k 个元素后执行删除步骤，WQS 中存储的 1 型元素数量至多为 2 型元素数量的 $O(\log t_k)$ 倍。

证明。 考虑 WQS 中一段最长的连续 1 型元素序列 $e_i, e_{i+1}, \ldots, e_j$，并且WQS中，b 值($e_i$) > b 值($e_{i+1}$) > ... > b 值($e_j$)。根据 $B^{(k)} = O(\log t_k$，带值的数量为公式 (13)。因此，该序列的长度最多为 $O(\log t_k)$。所以，不可能每个 2 型元素对应的 1 型元素数量都只有 $O(\log t_k$，证毕。 ∎

接下来，我们将重点通过以下引理来界定 2 型元素的数量。

引理 A.2. 在已看到 k 个元素后执行删除步骤，WQS 中存储的 2 型元素数量为 $O(\ell \log t_k)$。

证明。接下来，我们先给出 WQS 中每个 2 型元素 e_i 上 $G_i + g_{i+1}$ 的下界。

断言 A.3。对于任意 2 型元素 e_i，$G_{i+1} + g_i \geqslant 2^{\text{b-value}(e_{i+1})-1} - 2$。

证明。由于 e_i 是一个 2 型元素，因此有 $G_i+g_{i+1}+\Delta_{i+1} > t_k$。此外，我们知道 $\Delta_{i+1} \leqslant t_0(e_{i+1})$，由公式（17）可得。因此，

$$G_i + g_{i+1} > t_k - t_0(e_{i+1}) \geqslant 2^{\text{b-value}(e_{i+1})-1} - 2,$$

其中第二个不等式由定义 5.3得出。 ∎ 断言 A.3

我们将类型 2 元素划分为不同的集合。具体而言，我们考虑集合 $X_0, \ldots, X_{B^{(k)}}$，使得对于任意带值 α:

$$X_\alpha := \{e_i \in \texttt{WQS} \mid e_i \text{ is type-2 and b-value}(e_{i+1}) = \alpha\}.$$

注意， X_α 中的每个元素由于属于类型 2，因而也属于带$\leqslant\alpha$。对断言 A.3 中关于 X_α的每个元素的不等式求和，可得:

$$|X_\alpha| \cdot (2^{\alpha-1} - 2) \leqslant \sum_{e_i \in X_\alpha} G_i + g_{i+1} \leqslant 2 \sum_{e_i \in \texttt{WQS} \cap \text{Band} \leqslant \alpha} G_i, \tag{23}$$

其中，第二个不等式由以下事实得出： e_i 和 e_{i+1} 属于带$\leqslant\alpha$ ，并且 G_{i+1} 是 g_{i+1}的一个上界。证明的最后一步是给出公式 (23)右侧的一个界。具体而言，我们证明以下断言。

命题 A.4. 在观察到数据流中的 k 个元素后，对于任意 α, $\sum_{e_i \in \texttt{WQS} \cap \text{带}_{\leqslant\alpha}} G_i \leqslant O(\ell \cdot 2^{\alpha+1})$.

该命题与 命题 6.9 完全相同。证明只要求删除必须满足条件（i）（关于 b 值的条件）。在 算法 4 的删除步骤中，我们确实有相同的关于 b 值的条件。因此，证明与 命题 6.9 的证明完全相同。将 命题 A.4 中的界代入 公式 (23)，我们得到：

$$|X_\alpha| \cdot (2^{\alpha-1} - 2) \leqslant 2 \cdot O(\ell \cdot 2^{\alpha+1}).$$

因此，$|X_\alpha| = O(\ell)$，当 $\alpha > 2$ (且对于 $\alpha \leqslant 2$ 元素总数至多为 $8 \cdot \ell = O(\ell)$ 时，根据公式 (13) 无论如何皆是如此）。由于根据公式 (13)，带的可能取值只有 $O(\log t_k)$ 个，因此集合至多有 $\log t_k$) 个 X_α。由此可知，WQS 至多有 $O(\ell \cdot \log t_k)$ 个 2 型元素。 ∎

引理 A.1 和 A.2 直接推出，在经过 WQS中总共存储的元素数量，在 n次插入后为 $O(\ell \log^2 t_n) = O(\frac{1}{\varepsilon} \log^2(\varepsilon W_n))$，由此得到空间复杂度。

W我们先通过以下备注讨论 算法 4所用的空间。

备注 A.5。在空间分析中，我们将删除步骤后摘要中 2 型元素的数量界定为 $O(\ell \cdot \log t_n) = O((1/\varepsilon) \cdot \log(\varepsilon W_n))$，这本身相当高效。然而，如图 7 所示，在最坏情况下，每个 2 型元素可能对应 $O(\log t_n)$ 个 1 型元素。因此，算法 4 最终可能在摘要中存储多达 $O(\ell \cdot \log^2 t_n) = O((1/\varepsilon) \cdot \log^2(\varepsilon W_n))$ 个 1 型元素，导致其空间需求并非最优。

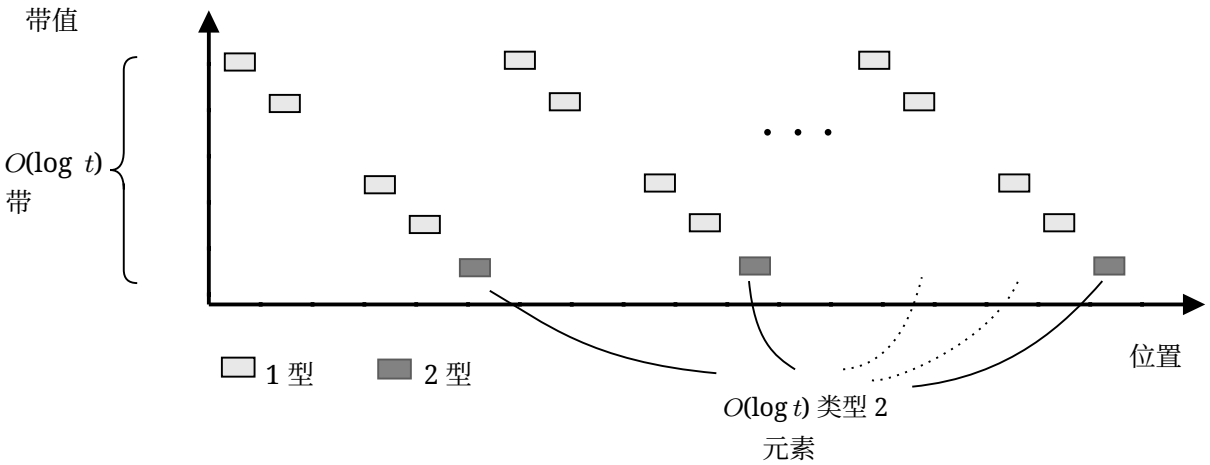


图 7：图中的每个块表示存储在 WQS 中的一个元素。元素的秩沿水平轴递增。该图说明了为什么 算法 4 最终可能会在 $O(\ell \log^2 t_n)$ 个元素存储在 WQS 中。根据 引理 A.2，$O(\ell \cdot \log t_n)$ 个 2 型元素可能存储在 WQS 中。每个这样的 2 型元素前面都可能有一串 $O(\log t_n)$ 个 1 型元素（因为有 $O(\log t_n)$ 个带）。

正如我们在 备注 A.5 中所说，算法 4 次优的一个原因，是摘要中存储的 1 型元素数量相对于 2 型元素过多。改进方法之一是主动尝试减少存储的 1 型元素数量。粗略来说，做法是仅在删除摘要中的 2 型元素不会导致形成一长串 1 型元素时才删除它（例如，图 图 7 所示）。注意，我们的算法 3 恰恰就是这样做的：只有在可以连同某个元素一起删除其整个段时，才删除该元素。

我们还要指出，即使我们没有在每次插入后都执行删除步骤，根据上述分析，执行删除步骤后，摘要的大小仍会缩减至 $O((1/\varepsilon)\log^2(\varepsilon W_n))$。因此，我们有如下备注。

备注 A.6 (延迟删除)。 假设在 算法 4 中，我们不再在每个元素之后运行第 (ii) 行中的 删除步骤 ，而是仅在插入 c 个元素 $c > 1$ 后才运行；那么，该算法的空间复杂度只会增加一个加性项 $O(c)$。

下面给出一种快速的实现，针对算法 4具体如下。

A.2 算法 4的高效实现

本节将介绍一种更快的算法 4实现，其基于（前文提到的）延迟删除步骤这一思路。

第一部分：存储分位数摘要：我们将分位数摘要 WQS 存储为一棵平衡二叉搜索树（二叉搜索树），其中每个节点包含 WQS 的一个元素及其元数据。对于每个元素 e，我们存储 $w(e)$ $g(e)$ $\Delta(e)$ 和 $t_0(e)$。二叉搜索树的排序键是元素的值。插入和删除操作分别将元素插入二叉搜索树以及从中删除元素。

第二部分：执行删除步骤：我们按如下方式执行此步骤：

1. 对 WQS（它是一棵二叉搜索树）进行中序遍历，得到一个按值排序的临时（双向链表）元素列表。

2. 使用定义 5.3计算WQS中所有元素的 b 值。

3. 从较大的元素到较小的元素遍历该列表。对于每个元素 e_i，如果它满足算法 4中提到的两个删除条件，则将其从二叉搜索树（以及列表）中删除。

请注意，遍历列表一遍后，不会再发生删除，因为我们是从较大的元素向较小的元素遍历列表，而删除操作只会增大下一个元素的 g 值。

现在，我们介绍一种 算法 4 的实现，其摊销更新时间较快。然后，我们将说明如何扩展该实现，使其最坏情况更新时间也较快。

实现 4：算法 4 的高效实现算法 4

- 将 WQS 初始化为空的平衡二叉搜索树。
- 删除时间 $\leftarrow 2$。
- 对于每个到达的项目（$x_k, w(x_k)$）：（i）运行插入（$x_k, w(x_k)$）。（ii）如果（k =删除时间）：– 执行删除步骤，并将删除时间更新为 $\leftarrow$ 删除时间 $+\ell\lceil\log^2 t_k\rceil$。

空间分析。 该实现的空间复杂度仍为 $O(\frac{1}{\varepsilon}\log^2(\varepsilon W_n))$。这是因为在删除步骤之后，当已插入 k 个元素时，只有再经过 $\lceil\ell\log^2 t_k\rceil$ 次插入后才会执行下一次删除。根据 备注 A.6，这意味着所用空间只会增加一个大小为 $O(\ell\log^2 t_k) = O(\frac{1}{\varepsilon}\log^2(\varepsilon W_n))$ 的加性项，因为 $t_k = O(\varepsilon W_n)$ 且 $\ell = O(1/\varepsilon)$。

时间分析。由于 WQS 存储为二叉搜索树，对 WQS 执行插入或删除操作需要$O(\log\ s)$，其中 s 是 WQS 中存储的元素数量。根据上述空间分析，$s = O(\frac{1}{\varepsilon}\log^2(\varepsilon W_n))$。由此得到以下观察：

观察 A.7。 对于长度为 n 的数据流，快速 实现 4 执行所有 插入 和 删除 操作所需的总时间为 $O(n\cdot(\log(1/\varepsilon) + \log\log(\varepsilon W_n))$。

该观察基于以下事实：每个元素在WQS中最多插入和删除一次，且每次插入或删除耗时为 $O(\log\ s) = O\big(\log(1/\varepsilon) + \log\log(\varepsilon W_n)\big)$。实现 4所花费的时间中，未计入观察 A.7的唯一部分，是确定要删除哪些元素所需的时间，我们将在下文给出其上界。

引理 A.8。对于长度为 n的数据流，实现 4在所有已执行的删除步骤中，判定需要删除哪些元素所耗费的总时间为 $O(n+\frac{1}{\varepsilon}\log^3(\varepsilon W_n))$。

证明。我们首先根据 t_k 的值，将 x_k S_w 中的元素划分为 $\lceil\log(\varepsilon W_n)\rceil$ 个不同的组：所有满足 $t_k\in[2^i,2^{i+1})$ 的元素 e_k 都属于同一组。令 $n(i)$ 表示数据流 e_k 中满足 $t_k\in[2^i,2^{i+1})$ 的元素数量。类似地，令 $d(i)$ 表示在 $t_k\in[2^i,2^{i+1})$ 时执行的删除步骤数量。两次连续删除之间，至少有 ℓi^2 个数据流元素被插入，因为在时间步 t_k 执行删除步骤后，我们会等待 $\ell\lceil\log^2 t_k\rceil$ 次插入。因此，区间 $[2^i,2^{i+1})$ 内删除步骤的数量满足以下界：

$$d(i)\leqslant\frac{n(i)}{\ell\cdot i^2}+1. \tag{24}$$

需要注意的一点是，在 $t_k\in[2^i,2^{i+1})$ 时，确定删除步骤中哪些元素需要删除所花费的时间与摘要的大小成正比： $O(s)=O(\ell\log^2 t_k)=O(\ell\cdot i^2)$。这是因为：创建链表并计算所有元素的 b 值需要 $O(s)$ 时间。最后，从最大元素到最小元素遍历列表（以检查是否满足删除条件）也需要 $O(s)$ 时间。

这与 公式 (24)，给出了在所有删除步骤中决定删除哪些元素所花费总时间的以下上界。

$$\begin{aligned}O\left(\sum_{i=1}^{\lceil\log(\varepsilon W_n)\rceil}d(i)\cdot\ell i^2\right)&=O\left(\sum_{i=1}^{\lceil\log(\varepsilon W_n)\rceil}\left(n(i)+\ell i^2\right)\right)\\&=O\left(n+\frac{1}{\varepsilon}\log^3(\varepsilon W_n)\right)\end{aligned}$$

至此，引理的证明完成。 ∎

观察 A.7 和引理 A.8 明显表明，对于长度为 n 的数据流，实现 4 所花费的总时间为 $O\big(n\cdot(\log(1/\varepsilon)+\log\log(\varepsilon W_n))+\frac{1}{\varepsilon}\log^3(\varepsilon W_n)\big)$。因此，每个元素的摊销更新时间为 $O\big(\log(1/\varepsilon)+\log\log(\varepsilon n)\big)$，当 $W_n=\text{poly}(n)$ 且 $\varepsilon\geqslant\log^3(n)/n$ 时。

最坏情况更新时间。 现在我们来考察任意元素的最坏情况更新时间。在最坏情况下，执行删除步骤时，我们可能需要从 $O(s)$ 个 分位数摘要 中删除元素，这将花费时间 $O(s\log s)$。为了降低最坏情况更新时间，我们对 实现 4作出一个小改动。我们注意到，下一次调用删除步骤是在 $\lceil\log^2 t\rceil$ 个时间步之后。为了降低最坏情况时间，我们将执行删除步骤所需的时间分摊到下一次删除步骤之前的所有元素上。形式化地说，我们有如下结论：

断言 A.9。 存在一种实现，可使 算法 4 在 $O(\log(1/\varepsilon)+\log\log(\varepsilon n))$ 最坏情况更新时间（按每个元素计算），其中 W_n 为 $\text{poly}(n)$，且 $\varepsilon\geqslant\log^3(n)/n$.

我们采用证明 断言 3.13时使用的相同标准技巧。其策略是在执行下一次删除步骤之前，将删除步骤均匀分摊到之后的所有时间步中。由于参数在 s方面相同，因此同样的论证也适用。这只会使空间增加一个常数因子，因此我们得以完成定理 4的证明。