arXiv:2302.06252v1 [cs.DS] 13 Feb 2023

游程编码字符串上的近最优动态时间弯曲

ItaiBoneh ！以色列赖希曼大学与海法大学
Shay Golan ✉以色列赖希曼大学与海法大学

Shay Mozes ✉以色列
赖希曼大学
OrenWeimann ！以色列海法大学

摘要

我们给出一个 $\tilde{O}(n^2)$ 时间算法，用于计算两个字符串之间的精确动态时间弯曲距离，这两个字符串的游程编码大小至多为 n。这与已知的（条件）下界相匹配（相差对数因子），并可与此前最快的 $O(n^3)$ 时间精确算法以及 $\tilde{O}(n^2)$ 时间近似算法进行比较。

2012年ACM学科分类 计算理论 → 模式匹配；计算理论 → 最短路径

关键词与短语 动态时间规整、弗雷歇距离、编辑距离、游程编码

资助 以色列科学基金会资助项目 810/21 .

1 引言

动态时间规整（DTW） [34] 是比较时间序列最受欢迎的方法之一（例如参见 [2,5,8,21,23,26,29,35,38]）。它在许多应用中都很有用，例如生物信息学、签名验证和语音识别；在这些应用中，两个时间序列的速度可能不同，但仍可被认为相似。例如，在语音识别中，即使一个人的说话速度比另一个人快，DTW 也能检测出相似性。

为定义 DTW，回想一下，字符串 S 在字母表 Σ 上的游程编码 $S = s_1^{\ell_1} s_2^{\ell_2} \cdots s_n^{\ell_n}$ 是对（长度为 $N = \sum_i \ell_i$ 的）字符串 S 的一种简洁表示（长度为 n）。这里 $s_i^{\ell_i}$ 表示字母 $s_i \in \Sigma$ 重复 ℓ_i 次。例如，字符串 $S = aaaabbbaaaaa$ 编码为 $a^4b^3a^5$。如果对每个 $\ell'_i \geq \ell_i$ 都满足条件，那么字符串 $S' = s_1^{\ell'_1} s_2^{\ell'_2} \cdots s_n^{\ell'_n}$ 就是字符串 $S = s_1^{\ell_1} s_2^{\ell_2} \cdots s_n^{\ell_n}$ 的时间扭曲。

▶ **定义 1**（动态时间规整）。对于函数 $\delta: \Sigma^2 \to \mathbb{R}^+$，字母表 Σ 上两个字符串 S 和 T 的动态时间弯曲距离定义为

$$\mathrm{DTW}(S,T) = \min_{|S'|=|T'|} \sum_{i=1}^{|S'|} \delta(S'[i], T'[i]),$$

其中，S' 和 T' 分别遍历 S 和 T 的所有时间扭曲。

1968 年，Vintzyuk [34] 提出了一种 $O(MN)$ 时间动态规划算法，用于计算两个字符串 S 和 T 的动态时间规整，它们的长度分别为 N 和 M。他的算法是最早使用动态规划的算法之一，如今仍在基础算法课程和教科书中讲授。除了对数因子改进之外，[15], 该算法的 $O(MN)$ 二次时间复杂度仍是已知最快的，而强次二次时间 $O((MN)^{1-\varepsilon})$

算法不太可能存在，因为这将推翻广为接受的强指数时间假设（SETH） [3, 9]。

关于 N 和 M，动态时间规整的复杂度已经得到了充分理解。动态时间规整的特殊情形也已得到充分研究，包括二进制字符串上的动态时间规整 [20, 33], 近似算法 [4, 19, 37], 大距离情形 [19], 稀疏输入 [17, 27, 28], 以及向其他相似性度量的归约 [19, 31, 32]。然而，就 n 和 m （分别为 S 和 T 的游程编码大小）而言，动态时间规整的复杂度尚未得到解决。也就是说，在字符串包含长游程这一（尤其具有吸引力的）情形下，目前最快的算法分别为 O（$Nm+Mn$） [12,13,19]和 O（n^2m+m^2n） [13]。特别是，只有在我们愿意接受（$1+\varepsilon$）-近似 [36]的情况下，才已知可以实现 $\tilde{O}$（nm）时间算法。˜能否获得精确的 O（nm）算法（除对数因子外达到最优）仍是一个悬而未决的问题。本文对这一开放问题给出了肯定回答。

关于动态时间规整的先前研究。 动态时间规整的经典动态规划如下。设 DTW(i, j) = DTW$(S[1 \dots i], T[1 \dots j])$，则 DTW$(0, 0) = 0$，对于所有 $i > 0$ 和 $j > 0$，均有 DTW$(i, 0)$ = DTW$(0, j) = \infty$，否则：

$$\mathsf{DTW}(i,j) = \delta(S[i], T[j]) + \min \begin{cases} \mathsf{DTW}(i-1, j) \\ \mathsf{DTW}(i, j-1) \\ \mathsf{DTW}(i-1, j-1) \end{cases} \tag{1}$$

上述动态规划等价于在以下网格图中进行单源最短路径（SSSP）计算。我们记 $[n] = \{1, 2, \dots, n\}$。

▶ **定义 2** (对齐图)。 S 和 T 的对齐图是一个有向带权图 G，其顶点为 $V = [0 \dots N] \times [0 \dots M]$。每个顶点 $(i, j) \in [N] \times [M]$ 有三条入边，权重均为 $\delta(S[i], T[j])$：一条从 $(i-1, j)$ 出发的垂直边，一条从 $(i, j-1)$ 出发的水平边，以及一条从 $(i-1, j-1)$ 出发的对角边。

我们将顶点 (0,0) 到 (i, j) 的距离记为 dist(i, j)。[1] 显然，DTW(i, j) =dist(i, j)。因此，DTW(S, T) =dist(N, M)，可通过单源最短路径算法在 $O(MN)$ 时间内计算得出（该算法会显式计算图中所有 $O(MN)$ 个顶点到 (0, 0) 的距离）。要优于 $O(MN)$，只需计算到部分顶点的距离。

也就是说，将对齐图划分为块，其中每个块都是对应于 S 中的一个游程以及 T中的一个游程的子图。然后，逐块处理，并针对每个块，根据其输入（上方块的最后一行和左侧块的最后一列）计算其输出（最后一行和最后一列）。由于块具有高度规则的结构（即块内所有边的权重都相同），因此可以在与输出大小成线性关系的时间内轻松计算输出。所有输出（以及所有输入）的总大小为 $O(Nm + Mn)$，因此可得到整体 $O(Nm + Mn)$ 时间算法 [12, 13, 19]。

为了低于 $O(Nm + Mn)$， [13] 中观察到，我们实际上并不需要计算完整的输出。只需计算输出与一组 $O(mn)$ 条对角线的交集即可。具体而言，每个块从其左上角贡献一条对角线，因此总共有 $O(mn)$ 条对角线，且每条对角线与 $O(m + n)$ 个块相交。这带来了一个 $O(n^2m + m^2n)$ 时间算法。最后， [36] 中证明，如果我们愿意接受 $(1 + \varepsilon)$-近似，那么只需计算 ˜ $O(1)$ 个输出值即可。

[1] 稍微滥用一下记号，我们之后也会用 dist$((x, y), (x', y'))$ 表示从顶点 (x, y) 到顶点 (x', y') 的距离。

关于编辑距离的先前工作。动态时间规整与编辑距离问题之间存在许多相似之处：(1) 与动态时间规整类似，编辑距离可以使用对齐图 [30, 34] 在 $O(MN)$ 时间内计算。唯一的区别在于边的权重。(2) 与动态时间规整类似，在 SETH [3, 9, 19], 条件下，编辑距离存在禁止强次二次时间算法的下界；(3) 与动态时间规整类似，可以通过逐块处理并根据输入计算输出来，在 $O(Nm + Mn)$ 时间内计算编辑距离。然而，不同于动态时间规整，已知可以在 $\tilde{O}(nm)$ 时间 [6, 7, 10–12, 16, 22, 24, 25] 内计算游程编码字符串的编辑距离。具体而言，Clifford 等人 [12] 证明，一个块的输入和输出可以通过分段线性函数隐式表示，并且可以从输入的表示 $\tilde{}$ 在摊销 $O(\text{polylog}(mn))$ 时间内计算出输出的表示。这意味着存在一个 $\tilde{O}(nm)$ 时间的编辑距离算法。

在 [36], Xi 和 Kuszmaul 谈到了获得 $O(nm)$ 时间算法来计算动态时间规整的前景：“这样的算法最终将统一游程编码设置下的编辑距离和动态时间规整”。

我们的结果与技术。我们提出了一种用于动态时间规整的 $O(nm)$ 时间算法。在 SETH 假设下，该算法在对数因子范围内是最优的。我们的算法与字母表大小 $|\Sigma|$ 以及函数 δ 无关。事实上，δ 甚至不必满足三角不等式。

我们沿用 Clifford 等人针对编辑距离提出的方法 [12]，使用分段线性函数来表示和处理输入与输出。然而，处理过程更具挑战性，原因有以下几点，Xi 和 Kuszmaul [36]曾对此进行阐述：(1) 与编辑距离不同，动态时间规整不满足三角不等式。(2) 我们关注的是适用于动态时间规整的任意代价函数 δ，而用于编辑距离的 $O(nm)$ 算法 [12]仅适用于 Levenshtein 距离（当 $\delta(\cdot,\cdot) \in \{0,1\}$ 时）。(3) 在标准设置（即非游程编码设置）下，编辑距离实际上可归约为动态时间规整 [19]。

在第2节中，我们证明，对输入和输出进行所需的操作可自然地归结为在一种数据结构上执行 $O(nm)$ 操作。给定一个大小为 A 的数组 $M + N$，该数组初始化为全零，并支持以下区间操作：

▶ 定义3（区间操作）。查找*(i)* - 返回 $A[i]$。*AddConst(i, j, c)* - 对每个 $k \in [i \dots j]$，设为 $A[k] \leftarrow A[k] + c$。*AddGradient(i, j, g)* - 对每个 $k \in [i \dots j]$，设为 $A[k] \leftarrow A[k] + k \cdot g$。*LeftLinearWave(i, j, α)* - 对每个 $k \in [i \dots j]$，设为 $A[k] \leftarrow \min_{t \in [i \dots k]} \big(A[t] + (k-t)\alpha\big)$。*RightLinearWave(i, j, α)* - 对每个 $k \in [i \dots j]$，设为 $A[k] \leftarrow \min_{t \in [k \dots j]} \big(A[t] + (t-k)\alpha\big)$。

在第3节中，我们展示主要的技术贡献：

▶**定理 4**。执行 s 定义 3 中的区间操作，摊销后每次操作的时间为$O(\text{多对数}(s))$ 。

定理的证明 4 大致可以描述如下：我们用A 的线性插值所形成的线段 A来表示数组。这样，定义 3 中的范围操作就转化为创建和删除线段、改变其斜率，以及将线段上下移位。对于大多数操作，这些变化只应用于 A 的一个连续范围，因此很容易在多对数时间内实现。较难处理的操作是LeftLinearWave 和RightLinearWave。这些操作可能需要将每个由 $\Omega(n)$个不同的连续线段组成的集合替换为一个新的线段。我们将用一个新的线段替换一组连续线段的过程称为射线发射过程。分别发射这些射线的代价会过高。更准确地说，用一个线段替换多个线段的射线发射过程并不成问题，因为其代价可以

计入线段数量的减少。难点在于发射射线，将单个线段替换为另一个线段，因为这不会减少线段数量。

我们的主要技术贡献是一种精巧的惰性方法，用于处理棘手的射线发射过程。我们研究射线发射过程的结构性质，并刻画了我们能够承担显式发射的长射线，以及无法承担显式发射的短射线。我们发现的结构使我们能够将表示 A 的线段划分为超级线段，并在每个超级线段中跟踪一条待处理短射线；这样，在每个超级线段中执行待处理射线发射过程，就能得到数组 A 的正确表示。虽然我们无法承担实际执行所有这些待处理的射线发射过程，但可以在局部执行该过程，例如，支持对 A 中特定元素的查找，或协助执行其他区间操作。

我们的惰性方法包含一个用于解决以下问题的数据结构（在编程奥林匹克竞赛中有时称为线段树 beats）：维护数组 A，支持查找查询以及两种更新：AddConst(i, j, c)——对每个 $k \in [i \dots j]$，设置 $A[k] \leftarrow A[k] + c$；以及 Min(i, j, c)——对每个 $k \in [i \dots j]$，设置 $A[k] \leftarrow$ 为 $\min\{A[k], c\}$。虽然我们不知道是否有正式发表的文献，但众所周知（例如参见 [1]），该问题可以在摊销多对数时间内解决。在第 4 节中，我们将展示一种不同的最坏情况多对数时间解法。[2]

2 通过区间操作实现动态时间规整

~本节证明，定理4蕴含一个 $O(nm)$ 动态时间规整算法。也就是说，动态时间规整可归约为高效支持定义3中的区间操作。

对齐图中的块。令 $S[i_1 \dots i_2]$ 和 $T[j_1 \dots j_2]$ 分别为 S 中的第 i 个游程和 T 中的第 j 个游程。对齐图中的块 $B_{i,j}$ 是由满足 $a \in [i_1 \dots i_2]$ 和 $b \in [j_1 \dots j_2]$ 的顶点 (a, b) 构成的集合。进入块 $B_{i,j}$ 中任意顶点的所有边权重相同，为 $\delta(S[i_1], T[j_1])$，记作 $c_{B_{i,j}}$。我们将块 $B_{i-1,j}$、$B_{i,j-1}$ 和 $B_{i-1,j-1}$ 称为 $B_{i,j}$ 的进入块。块的输入由属于该块第一行或第一列的所有顶点组成。块的输出由属于该块最后一行或最后一列的所有顶点组成。以往研究也曾隐含地使用了以下结构引理（正式证明见附录）。

▶**引理 5.** 设 B 是一个块。

- 如果 (x, y)、$(x, y+1) \in B$ 那么从 $(0, 0)$ 到 $(x+1, y+1)$ 存在一条不经过 $(x, y+1)$ 的最短路径。如果 (x, y)、$(x+1, y) \in B$ 那么从 $(0, 0)$ 到 $(x+1, y+1)$ 存在一条不经过 $(x+1, y)$ 的最短路径。如果 (x, y)、$(x+1, y+1) \in B$ 那么从 $(0, 0)$ 到 $(x+1, y+1)$ 存在一条经过 (x, y) 的最短路径。

对齐图中的前沿。 我们的动态时间规整算法会处理对齐图中的所有块。在每一步中，只要某个块 B 的所有进入块都已处理，算法就可以处理该块。当块 B 被处理时，算法

[2] 我们注意到，[1] 该方案还支持区间和查询，而对于这类问题，已知存在一个条件下界（来自在线矩阵向量乘法（OMV）问题）[14]。该下界表明，最坏情况下的 操作不太可能在 $O(n^{1/2-\varepsilon})$ 时间内完成。我们能够规避这个下界，因为我们只支持查找，而不支持区间和查询。

会为 B的每个输出顶点（x， y）计算 dist(x， y)。处理完块 B后，我们称 B 的输出顶点已确定。算法的每一步中，前沿 是这样一组已确定的顶点：它们各自有一条出边指向尚未处理的块。注意，在算法执行的任意时刻，对于每个值 $d \in [-N \dots M]$，前沿中恰好包含一个满足 $y - x = d$的顶点（x， y）。在算法的每一步 t 中，我们将维护一个数组 F $_t[-N \dots M]$，其中 $F_t[d] =$dist(x， y)，且顶点（x， y）属于当前前沿并满足 $y - x = d$。本节的主要结果是以下引理：

▶**引理 6.** F_{t+1} 可通过对 $O(1)$ 执行区间操作（定义 3）得到。 F_t。

在证明引理 6之前，我们先证明它蕴含了主要结果：

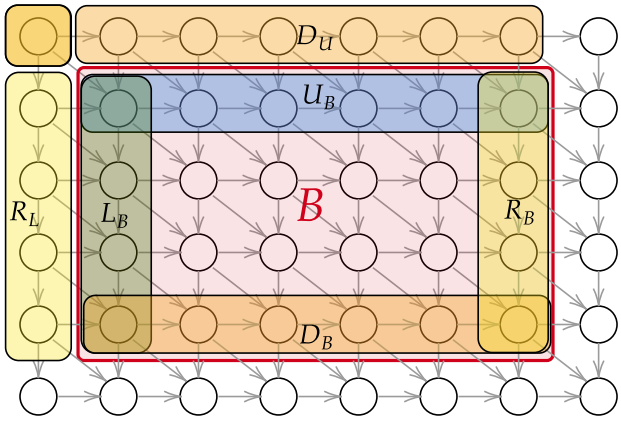

图 1 块 B、其输入 $L_B \cup U_B$ 和输出 $D_B \cup R_B$。进入 L_B的边来自 R_L、 C的角以及 D_U最左侧的节点。进入 U_B 的边来自 D_U、 C的角以及 R_L最顶部的节点。

▶ **定理 7.** 两个游程编码字符串 S 的动态时间弯曲距离 a ˜和 T 分别包含 n 和 m 个游程时，可在 $O(nm)$ 时间内计算。

证明。我们将定理 4 的数据结构初始化为一个长度为 $N + M + 1$ 的数组。我们将 A 的索引视为处于 $[-N \dots M]$[3] 中。初始时，前沿由顶点 (x, 0)（满足 $x \in [0 \dots N]$）和 (0, y)（满足 $y \in [M]$）组成。我们首先将 A 转换为 F$_0$。根据公式（1），我们需要为 $i \neq 0$ 设置 $A[0] = 0$ 和 $A[i] = \infty$。这可以通过执行 AddConst(1, M, ∞) 和 AddConst($-N$, -1, ∞) 来完成。

该算法运行 nm 次迭代。在第 t 次迭代开始时，我们有 $A =$ F$_t$。算法任意选取一个进入块已处理完毕的块 B，并在 A 上执行O(1) 次区间操作（依据引理 6），以获得 $A =$ F$_{t+1}$。最后一次迭代结束后，可以保证块 $B_{n,\ m}$ 已经处理完毕。因此，F$_{nm}[M - N]$ =动态时间规整(S， T)。每次迭代都需要执行 O(1) 次区间操作，每次耗时 O(多对数(nm))，因此˜该算法总共执行 $O(nm)$ 次操作，耗时总计 $O(nm)$。◀

在本节的其余部分，我们证明引理 6。

[3] 当梯度更新 AddGradient(i， j， c) 影响值 $A[k]$ 时，我们希望 $A[k]$ 增加 $k \cdot c$，其中 $k \in [-N \dots M]$ 是“模拟”索引，而非实际索引 $k + N + 1$。这可以通过应用额外操作 AddConst(i， j，$(-N - 1) \cdot c$) 来实现。

概述。我们分两个阶段从 F_t 得到 F_{t+1}。设 B 为在步骤t处理的块，并假设 B对应于游程 $S[i_1 \ldots i_2]$和 $T[j_1 \ldots j_2]$。那么，F_t与 F_{t+1}仅在范围 $[a \ldots b]$内有所不同，其中 $a = j_1 - i_2$和 $b = j_2 - i_1$（见图 2）。在第一阶段，我们对 F_t 执行一系列区间操作，以得到 F；F 被定义为与 F_t 完全相同，但 B 的输入替换了其进入块 B的相应输出。形式化地，对于每个 $d \notin [a \ldots b]$，$\mathrm{F}[d] = \mathrm{F}_t[d]$；而对于 $d \in [a \ldots b]$，$\mathrm{F}[d] = \mathsf{dist}(x, y)$，其中（$x$，$y$）是具有 $y - x = d$的 B的输入节点。在第二阶段，我们对 F 执行另一系列区间操作，以得到 F_{t+1}；F_{t+1}与 F 完全相同，但 B 的输入被 B的输出所替换。

B的高度记作 $h = i_2 - i_1 + 1$，B的宽度记作 $w = j_2 - j_1 + 1$。我们将 B 的第一行记作 U_B，第一列记作 L_B；将 B 的最后一行记作 D_B，最后一列记作 R_B（见图 1）。我们注意到，B 的输入节点为 $L_B \cup U_B$，输出节点为 $D_B \cup R_B$。我们将 B 的进入块记作 $L = B_{i-1,\, j-1}$、$C = B_{i-1,\, j-1}$ 以及$U = B_{i-1,\, j}$。我们将 L的最后一列定义为 R_L，将 U的最后一行定义为 D_U。注意，R_L的顶点对应的 $\mathsf{dist}(x, y)$ 值存储在 $\mathrm{F}_t[a-1 \ldots a+h-2]$中，而 D_U的顶点对应的 $\mathsf{dist}(x, y)$ 值存储在 $\mathrm{F}_t[b-w+2 \ldots b+1]$ 中（见图 2）。

第一阶段——从 F_t 计算 F。我们首先计算 $\mathsf{dist}(i_1, j_1)$。回顾一下，$\mathsf{dist}(i_1 - 1, j_1 - 1)$ 存储在 $\mathrm{F}_t[z]$中，其中 $z = a + h - 1 = b - w + 1 = j_1 - i_1$。根据公式（1），我们有 $\mathsf{dist}(i_1, j_1) = c_B + \min\{\mathrm{F}_t[z-1], \mathrm{F}_t[z], \mathrm{F}_t[z+1]\}$。令 F_t为 F_t，并将 $\tilde{\mathrm{F}}_t[z] \leftarrow$ 赋值为 $\mathsf{dist}(i_1, j_1) - c_B$。F 的定义由以下引理提供依据。

▶ **引理 8.** 对于每个 $k \in [z \ldots b]$，$\mathrm{F}[k] = c_B + \mathrm{Min}_{i \in [z \ldots k]} \left(\tilde{\mathrm{F}}_t[i] + (k - i)c_B \right)$。

证明。对于 $k \in [z \ldots b]$，令 $y = k - z$。注意，应将 $\mathrm{F}[k]$ 赋值为 $\mathsf{dist}(i_1, j_1 + y)$。我们通过对 y 进行归纳法来证明该断言。对于 $y = 0$，我们需要证明 $\mathrm{F}[z] = \mathsf{dist}(i_1, j_1) = c_B + \mathrm{F}_t[z]$。这由以下事实得出：$\mathrm{F}_t[z] = \mathsf{dist}(i_1, j_1) - c_B$。对于归纳步骤，我们需要证明 $\mathrm{F}[k] = \mathsf{dist}(i_1, j_1 + y) = \min_{i \in [z \ldots k]} \left(\tilde{\mathrm{F}}_t[i] + (k - i)c_B \right) + c_B$。

根据引理 5，有两种情况：(i) 到 $(i_1, j_1 + y)$ 的最短路径沿对角线经过 $(i_1 - 1, j_1 + y - 1)$。那么，根据 F_t 的定义，其长度为 $\mathsf{dist}(i_1 - 1, j_1 + y - 1) + c_B = \mathrm{F}_t[k] + c_B$。(ii) 到 $(i_1, j_1 + y)$ 的最短路径水平经过 $(i_1, j_1 + y - 1)$，因此其长度为 $c_B + \mathrm{F}[k-1]$。那么，根据归纳假设，这条路径的长度为

$$\left(c_B + \min_{i \in [z \ldots k-1]} \left(\tilde{\mathrm{F}}_t[i] + ((k-1) - i)c_B \right) \right) + c_B = \min_{i \in [z \ldots k-1]} \left(\tilde{\mathrm{F}}_t[i] + (k - i)c_B \right) + c_B.$$

取 (i) 和 (ii) 中的最小值即可得到该引理。 ◀

由引理 8 可直接得出，$\mathrm{F}_t[z \ldots b]$ 可以通过依次应用以下区间操作$\mathrm{F}[z \ldots b]$ 转换为：

1. $\mathsf{AddConst}(z, z, \text{距离}(i_1, j_1) - c_B - \mathrm{F}_t[z])$。2. $\mathsf{LeftLinearWave}(z, b, c_B)$。3. $\mathsf{AddConst}(z, b, c_B)$。

第一个操作将 F_t 变为 F_t，另外两个操作则通过应用引理 $\mathrm{F}_t[z \ldots b]$ 中的公式将其变为 F。类似地，我们可以证明 8。可以从 $\mathrm{F}[a \ldots z]$ 得到 F_t。这一次，使用 $\mathsf{RightLinearWave}(i, j, c)$。

第二阶段 - 计算 F_{t+1} 从 F。 下面的引理将说明，F_{t+1} 可以通过对 F 应用 $O(1)$ 次区间操作得到。

▶ **引理 9**。 设 $d_1 = j_1 - i_1$ 和 $d_2 = j_2 - i_2$。对于每个 $d \in [a \dots b]$:

$$\mathsf{F}_{t+1}[d] = \begin{cases} \mathsf{F}[d] + (d-a)c_B, & \text{if } d \in [a \dots \min(d_1, d_2)) \\ \mathsf{F}[d] + \min(w,h)c_B, & \text{if } d \in [\min(d_1, d_2) \dots \max(d_1, d_2)] \\ \mathsf{F}[d] + (b-d)c_B, & \text{if } d \in (\max(d_1, d_2) \dots b] \end{cases}$$

证明。 我们先给出以下断言:

▷ 断言 10. 设 (x, y) 是块 B 中的一个顶点，且 (x', y') 是 B 的输入顶点，满足 $y' - x' = y - x$，则 $\text{dist}(x, y) = \text{dist}(x', y') + c_B \cdot (x - x')$。

证明。我们对 $d = x - x'$ 进行归纳，证明存在一条从 $(0, 0)$ 到 (x, y) 的最短路径，该路径经过 (x', y')，然后连续使用 $x' - x$ 条对角边。若 $d = 0$，则结论显然成立。否则，由于 $d \geq 1$，顶点 (x, y) 和 $(x-1, y-1)$ 均位于同一个块 B 中。由引理 5 可知，存在一条经过 $(x-1, y-1)$ 到达 (x, y) 的最短路径。根据归纳假设，该路径经过 (x', y')，随后仅使用对角边。◀

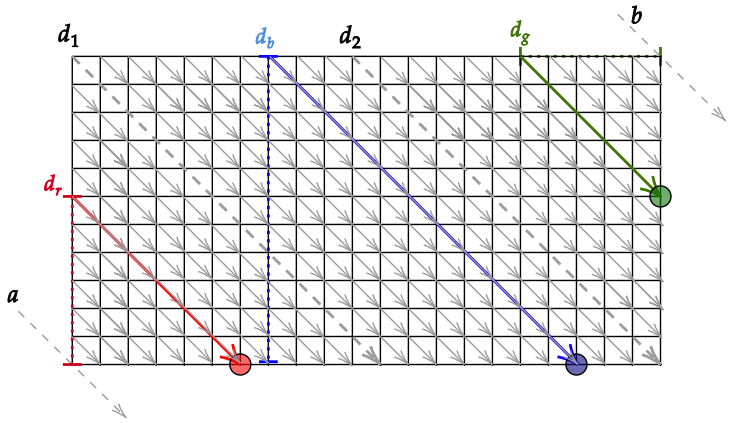

图 2 一个块 B。红色顶点是对角线 d_r 上的一个输出顶点，其 $d_r \in [a \dots d_1]$为 $d_r - a$，并且从同一对角线上的匹配输入到红色顶点的对角步数为 $d_r - a$。类似地，蓝色顶点位于对角线 $d_b \in [d_1 \dots d_2]$上，绿色顶点位于对角线 $d_g \in [d_2 \dots b]$上。

设 (x, y) 是 B 在对角线 $d = y - x$ 上的一个输出。设 (x', y') 是 B 在同一对角线 d 上的输入。因此，以下结论成立（见图 2）：

1. 如果 $d \in [a \dots \min(d_1, d_2))$，则有 $x - x' = d - a$。 2. 如果 $d \in [\min(d_1, d_2) \dots \max(d_1, d_2)]$，则有 $x - x' = \min(w, h)$。 3. 如果 $d \in (\max(d_1, d_2) \dots b]$，则有 $x - x' = b - d$。

结合断言 10，即可证明引理 9。 ◀

根据引理 9，F 可以通过应用以下操作转换为F_{t+1}：
1. AddConst(a, $\min(d_1, d_2) - 1, -a \cdot c_B$). 2.
AddGradient(a, $\min(d_1, d_2) - 1, c_B$). 3.
AddConst($\min(d_1, d_2)$,$\max(d_1, d_2)$, $\min(w, h)c_B$). 4.
AddConst($\max(d_1, d_2) + 1, b, b \cdot c_B$). 5.
AddGradient($\max(d_1, d_2) + 1, b, -c_B$).

至此，第二阶段以及引理 6 的证明告一段落。至于所需范围查询 d_1、d_2、w、h、a、b、c_B，以及 z的参数，都可以用直接的方法预先计算出来，且适用于每个块，耗时为 B在$O(nm)$时间。

3 实现区间操作

在本节中，我们证明定理 4。我们将数组 A 视为分段线性函数。对于 A，我们关联一个点集 P $=\{p_1=(x_1,y_1)\ p_2=(x_2,y_2)\ldots\}$，其中的点满足 $A[x_i]=y_i$。集合 P 由 A 唯一确定，它们是 A的线性插值中最大线性线段的端点。注意，P 的第一个点始终为 $(1,A[1])$，最后一个点为 $(n,A[n])$。[4] 令 $\ell_i(x)=\alpha_i x+\beta_i$ 表示连接 p_i 和 p_{i+1}的线段。我们的表示将维护这些 α_i和 β_i。借助这种表示，我们可以根据 α_i和 β_i，检索任意 $x\in[1,n]$处的 $A[x]$，其中 x_i是序列 $(x_1,x_2,\ldots)$中 x的前驱。初始化时，A 表示为一条线性线段，其中 $\alpha_1=0$，且 $\beta_1=0$。

我们将使用以下简单的数据结构 re.[5]

▶**引理 11** (区间加法数据结构)。有一种数据结构，可对一组首坐标互不相同的 n 点支持以下操作，每次操作耗时为 $O(\log n)$ 。

- 查找*(x)* - 若存在首坐标为 x 的点，则返回该点的第二坐标。插入*(x, y)* - 插入点 *(x, y)*。
- 删除*(x)* - 若存在首坐标为 x 的点，则删除该点。*AddToRange(i, j, c)* - 对每个满足
- $x\in[i\ldots j]$ *set* $y\leftarrow y+c$ 的点 *(x,y)* 执行操作。下一个大于点*(x, y)* - 返回点 $p'=(x',y')$，
- 其 $x'>x$ 在满足 $y'>y$ 的点中最小。上一个小于点*(x, y)* - 返回点 $p'=(x',y')$，其
- $x'<x$ 在满足 $y'<y$ 的点中最大。

3.1 预热算法

我们首先给出一种用于区间操作数据结构的朴素且低效的实现。我们在定义于序列 $(x_1,x_2,\ldots)$ 上的前驱 / 后继数据结构中维护序列 P。稍微滥用一下记号，我们也用 P 指代这个数据结构。我们分别使用两个区间加数据结构 D_α 和 D_β来维护 α_i 和 β_i。从 x_i 开始的线性线段 ℓ_i 的参数 α_i,β_i，分别用 D_α 中的点 (x_i,α_i) 和 D_β 中的点 (x_i,β_i) 表示。下文中，每当我们说向 P 添加一个点 $p=(x,y)$，意思是将 (x,y) 插入前驱/后继数据结构 P，并将第一坐标为 x 的点插入 D_α 和 D_β，同时适当设置它们的第二坐标，以反映终止于 p 的线段和始于 p 的线段的参数 α,β。这一过程需要在 P D_α 和 D_β 上执行 $O(1)$ 次操作。

AddConst(i,j,c) 的作用（见图3）是将包含 i 的线段拆分为至多 3 条线性线段（一条终止于 $i-1$ 的前缀、一条线段 $[i-1,i]$，以及一条始于 i 的后缀），对包含 j 的线段也进行类似处理。因此，要应用 AddConst(i,j,c)，我们首先通过在 P D_α 中插入或更新线段的端点，将包含 i 和 j 的线段替换为这些 $O(1)$ 条新线段，然后 D_β。接着，我们在 D_β 上调用 AddToRange(i,j,c)，将 i 和 j 之间的所有线段移位 c。接下来，我们为线段 $[i-1,i]$ 和线段 $[j,j+1]$ 设置参数，方法是 O 次额外调用 AddToRange，对 D_α 和

[4] 这里我们用 n 表示数组 A 的大小。[5] 该数据结构可以用带有增量表示的平衡搜索树来实现（其中节点的值表示为其所有祖先节点的值之和），并且让每个节点也存储其子树中的最小值和最大值。参见例如 [18]。

D_β。最后，我们检查插入的任何新线段是否与其相邻线段具有相同的斜率；如果是，则从 $\mathcal{P}$ D_α 和 D_β 中移除它们的公共点，将这些线段合并为一条线段。这确保我们维护的集合 $\mathcal{P}$ 确实是 A 定义的集合 $\mathcal{P}$。支持 AddGradient(i, j, g) 的方法类似。唯一的区别是，我们在 D_α 上调用 AddToRange(i, j, g)，而不是在 D_β 上调用，因为发生移位的是线段的斜率，而不是它们的值。

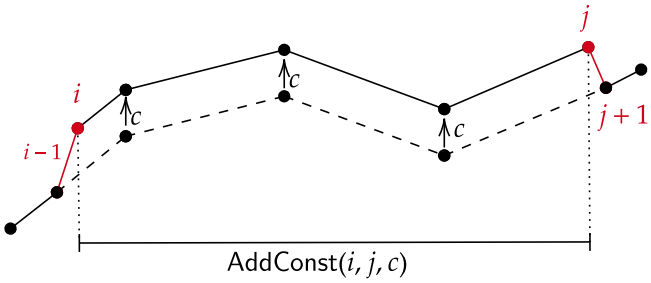

图3 展示了执行 AddConst(i, j, c) 操作的示意图。虚线表示操作前的线段。操作后，会创建带有 x 坐标$i-1$, i, j的点，而 $[i \dots j]$ 中的线段会移位 c。

因此，难点在于支持 LeftLinearWave(i, j, α)。我们先描述它的效果，再说明如何实现。我们不失一般性地假设 i 和 j 都是线段的端点（否则，我们会像上文那样将包含它们的线段拆分为 $O(1)$ 个线段）。令 $p_a = (i, A[i])$ 和 $p_{b+1} = (j, A[j])$ 分别为对应于 i 和 j 的点。因此，包含在 $[i \dots j]$ 内的线段为 ℓ_a, $\ell_{a+1} \dots \ell_b$。

若 $\alpha_a \le \alpha$，则线段 ℓ_a 不受线性波影响。这是因为对于每个 $k \in [x_a \dots x_{a+1}]$，线性波赋值为 $A[k] \leftarrow \min_{x_a \le t \le k}(A[t] + (k-t)\alpha) =$

$$= \min_{x_a \le t \le k}(A[k] - (k-t)\alpha_a + (k-t)\alpha) = \min_{x_a \le t \le k}(A[k] + (k-t)(\alpha - \alpha_a)) = A[k].$$

令 $z \in [a \dots b]$ 为满足 $\alpha_z > \alpha$的最小索引。根据相同的推理，线段 ℓ_a、ℓ_{a+1}、……、ℓ_{z-1} 均不受线性波影响。令 $r_z(x)$为一条斜率为[6] α 、起点为 p_z的（正）射线。由于 $\alpha_z > \alpha$，射线 r_z 位于线性线段 ℓ_z下方。因此，起点为 p_z 的线段 ℓ_z 会受到线性波影响；其斜率从α_z 变为 α，并且只要 $A[x] \ge r_z(x)$，就会延伸至 x_{z+1} 之外。我们通过从 p_z 进行射线发射过程来描述 LeftLinearWave 的这一作用（见图4）。该过程按如下方式确定 ℓ_z 的新端点 p'，并删除 p_z 与 p'之间所有现有的线段。

令 $z' \in [z+1..b+1]$ 为满足 $y_{z'} < r_z(x_{z'})$ 的最小下标，即 $\mathcal{P}$ 中严格位于射线 r_z 下方的第一个点。令 $p^* = (x^*, y^*)$ 为射线 r_z 与 $\ell_{z'-1}$ 的交点（如果 z' 不存在，则 $p^* = p_b$）。ℓ_z 的新端点是点 $p' = (x', y') = (\lfloor x^* \rfloor, r_z(\lfloor x^* \rfloor))$，并替代所有满足 p_w 的点 $w \in (z \dots z')$。如果 x^* 不是整数（或者 z' 不存在），则在 p' 和 $p'' = (x'+1, A[x'+1])$ 之间形成一条新的线段。

以同样的方式分析 LeftLinearWave(i, j, α) 对 A 剩余部分的影响，也就是对 $A[x_{z'} \dots j]$ 的影响；这次从 $p_{z'}$ 而非 p_a 开始。在

[6] 注意，在图4 以及后续所有图中，我们通过绘制射线与 x轴正方向之间的角 α 来表示射线 r_z 的斜率 α。然而，形式上 α 是射线的斜率，而非所示的角。

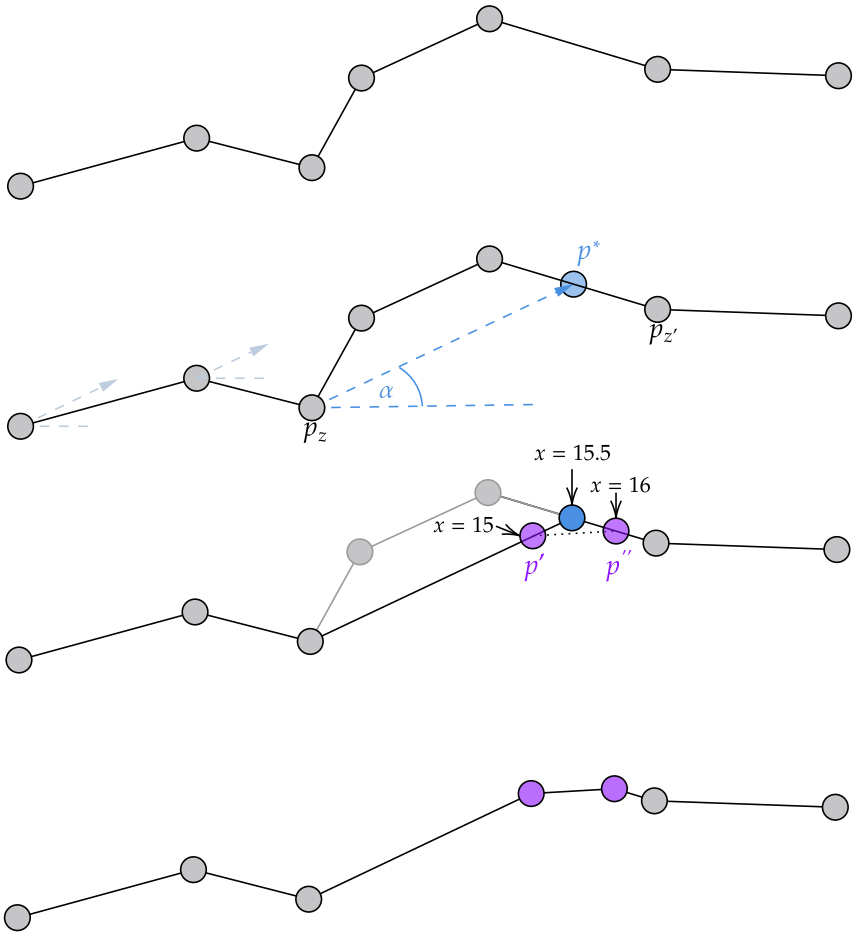

图4 LeftLinearWave(i, j, α) 的效果。 p_z 之前的线段不受影响。 p_z 与 $p_{z'}$ 之间的线段受到影响。具体而言，从 p_z 发出一条斜率为 α 的射线 r_z（蓝色虚线），并在点 $p^* = (x^*, y^*)$ 处相交。 ℓ_z 的新端点变为 p'，而 p_z 与 p' 之间的所有线段都被删除。由于 $x^* = 15.5$ 不是整数，因此在 p'（坐标为 x，即 15）与 p''（坐标为 x，即 16）之间形成一条新线段。

特别是，斜率小于 α 的线段前缀不受影响；从下一个 p_w 发出一条斜率为 α 的射线，且 $\alpha_w > \alpha$，依此类推。在附录（引理 23）中，我们将正式证明，上述刻画确实表示了 $A[i \dots j]$ 的新值。

现在，我们根据上述描述，介绍 LeftLinearWave(i, j, α) 的一种朴素、低效实现。回想一下， i 和 j 被假定为线段的端点 p_a 和 p_{b+1}。我们首先通过查询 D_α.nextGT(i, j, α)，找到第一个满足 $x_z \in [i \dots j]$ 和 $\alpha_z > \alpha$ 的 p_z。然后从 p_z（如果 p_z 存在）执行如下射线发射过程：回想一下， $r_z(x)$ 表示从 p_z 发出、斜率为 α 的正向射线。我们按顺序逐个扫描 p_z 的后继点，并对每个点 p_w 检查射线 $r_z(x_w) \le y_w$ 是否成立。如果成立，则通过从 P 中删除 x_w，来删除 p_w， D_β 和 D_α。否则，我们计算 $p^* = (x^*, y^*)$，即 r_z 与 ℓ_{w-1} 的交点，并据此计算点 $p' = (x', y')$、 $= (\lfloor x^* \rfloor, r_z(\lfloor x^* \rfloor))$，如果 x^* 不是整数，还要计算 $p'' = (\lceil x^* \rceil, \ell_{w-1}(\lceil x^* \rceil))$。接着，如上文对 AddConst 的讨论，在 p_w 之前插入新点 p' 和 p''。随后继续扫描，并从 p_w 开始再次查询 nextGT，如此继续。若在过程结束时

由于最后一个点 p_b 位于某个 r_z 上方而被删除，我们就插入一个新点 $(x_b, r_z(x_b))$。

现在我们分析这一朴素实现的时间复杂度。每次 AddConst 和 AddGradient 操作都需要在区间加数据结构上执行 $O(1)$次操作，因此每次操作耗时为 $O(\text{polylog}(|\mathcal{P}|))$，其中 $|\mathcal{P}|$ 是执行该操作时$\mathcal{P}$ 的基数。

对于 LeftLinearWave 操作，我们可能会希望将每次射线发射过程的开销计入过程中从$\mathcal{P}$ 移除点的代价。然而，每次射线发射过程也可能添加至多两个新点，从而导致$\mathcal{P}$ 的大小增加。事实上，一次 LeftLinearWave 操作可能引发许多这样的射线发射过程，因此可能大幅增加$\mathcal{P}$ 的大小并耗费过多时间。这是我们需要解决的主要技术难题。

我们的思路是区分长 射线发射和短 射线发射：前者可以将新插入全局计费，后者则不能。我们按照朴素解法处理长射线，并设计一种单独的延迟机制，通过对一个用于跟踪延迟射线的独立数据结构执行常数次更新，推迟应用同一次 LeftLinearWave 操作所产生的所有短射线。

对称性：RightLinearWave。 目前为止的讨论都集中在 LeftLinearWave 操作上。我们注意到，对 RightLinearWave 的分析与此对称。具体而言，RightLinearWave(i, j, α) 的执行过程可以描述为一系列沿负向 射线进行的射线发射。最先发射射线的点是满足 $\alpha_{z-1} < -\alpha$ 的最大 $z \in [a \dots b]$ 所对应的 p_z（p_z 通过 D_α.上一个小于点 找到）。请注意，RightLinearWave 开始射线发射过程的条件涉及 α_{z-1}，而非 α_z，因为 p_z 左侧线段的斜率为 α_{z-1}。为简化表述，我们将继续只描述 LeftLinearWave，并在最后简要说明为处理对称的 RightLinearWave 所需进行的调整。

3.2 活动点与被动点、长射线与短射线

在正式定义长射线和短射线之前，我们首先注意到，只有在斜率增大的点处才会进行射线发射。我们称这类点为活跃 点。

▶ **定义 12**（活动点与被动点）。 如果点 p_z 满足 $z \in \{1, |P|\}$ 或 $\alpha_z > \alpha_{z-1}$，则称其为活跃。非活跃的点称为被动。我们用 P_{active} 表示活跃点集合。

▶ **引理 13**。 由LeftLinearWave（i, j, α）发出的射线，要么在点$p_a = (i, A[i])$ 处发出，要么在活跃点处发出。

证明。 反设射线发射过程始于一个被动点 $p_z \neq p_a$。如果 p_z是射线发射开始的第一个点，那么 z就是 $[a \dots b]$中满足 $\alpha_z > \alpha$的最小索引。但由于 p_z 是被动点，因此有 $\alpha < \alpha_z \leq \alpha_{z-1}$，这与 z的最小性矛盾（注意， $p_z \neq p_a$因此 $z - 1 \in [a \dots b]$）。

否则，令 p_q 为在 p_z 之前最后一个发生过射线发射过程的点。令 $p_{q'}$ 为射线 p_q 下方的第一个点。由于 p_z 是下一个发射射线的点，因此 z 是 $[q \dots b]$ 中满足 $\alpha_z \geq \alpha$ 的第一个点。由于 p_z 是非活跃的，我们有 $\alpha < \alpha_z \leq \alpha_{z-1}$。若 $z \neq q'$，则有 $z - 1 \in [q' \dots b]$，这与 z 的最小性矛盾。否则， $p_z = p_{q'}$ 是从 p_q 发射的斜率为 α 的射线下方的第一个点。因此， p_{z-1} 在射线上方，且 $\alpha_{z-1} > \alpha$。必定是 $p_{q'}$ 在射线上方，这就产生了矛盾。

◀

12 游程编码字符串上的近最优动态时间规整

同样，我们在附录 A 中为关于以下内容的对称命题给出RightLinearWave证明

▶ **引理 14.** 源自 RightLinearWave(i, j, α) 的射线发射，要么发生在点$p_b = (j, A[j])$ 处，要么发生在活动点处。

令 $\mathcal{P}_{\text{active}} = q_1, q_2 \ldots$ 为序列 $\mathcal{P}$ 在活动点上的限制。我们可以将活动点视为定义了一个分段线性函数，其线段是对 A 中线段的粗化。我们将这些线段称为超级线段。令 γ_z表示端点为 q_z 和 q_{z+1}的超级线段的斜率。以下引理断言， A 的线段始终不会低于其对应的超级线段，并且从活动点开始的线段斜率不会小于从同一点开始的超级线段斜率。

▶ **引理 15.** 设 $q_z = p_w$ 和 $q_{z+1} = p_{w'}$ 为两个相邻的活动点。对于每个$k \in [w \ldots w']$，被动点 p_k 不会低于连接 q_z 和 q_{z+1}的超级线段。此外， $\alpha_w \geq \gamma_z$。

证明。 （见图 5）反设存在一个位于超线段下方的点，并令 $k' \in (w \ldots w')$ 为这类点中索引最小的点。由于 $p_{k'-1}$ 不在超线段下方，而 $p_{k'}$ 在超线段下方，因此必有 $\alpha_{k'-1} < \gamma_z$。此外，由于满足 p_k 且 $k \in [k' \ldots w')$ 的点均为非活跃点，斜率非递增，因此每个$\alpha_k \leq \gamma_z$。这意味着所有这些点，尤其是 $p_{w'}$ ，都位于超线段下方。这与 $p_{w'}$ 位于超线段上矛盾。 ◀

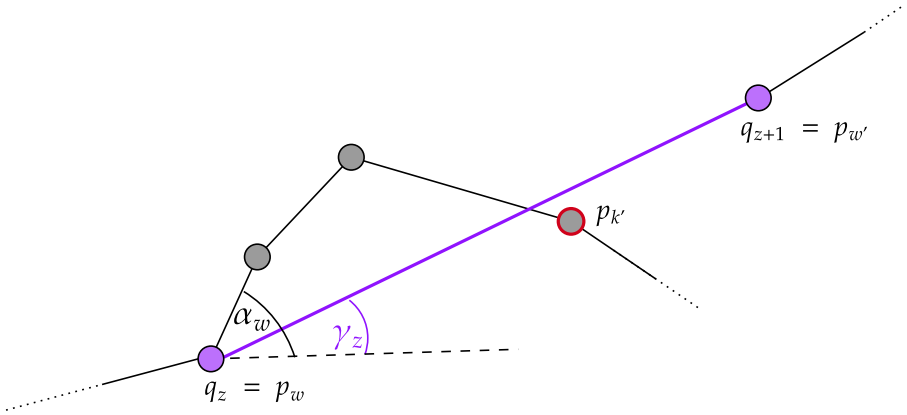

图 5 引理 15中的不可能构型。点 q_z 和 q_{z+1} 用紫色点表示，连接它们的超线段用粗紫线表示。线段下方的第一个点 $p_{k'}$ 以红色描边标出。由于超线段严格内部的点均为非活跃点，因此位于超线段内、紧随 $p_{k'}$ 之后的点（尤其是 q_{z+1}）必定仍位于超线段下方。

接下来我们证明，若射线发射过程从一个满足 $\gamma_z < \alpha$ 的活动点 q_z 开始，则该过程会在 q_{z+1} 之前结束，且唯一受到影响的点是 q_z 与 q_{z+1} 之间的被动点。另一方面，若 $\gamma_z \geq \alpha$ ，则该过程会导致 q_{z+1} 不再是活动点，因此 $\mathcal{P}_{\text{active}}|$ 会减少。

▶ **引理 16。** 考虑在应用 LeftLinearWave(i, j, α)期间从点 $p_w = q_z \in P_{\text{active}}$ 开始的射线发射过程。设 $q_{z+1} = p_{w'}$。

1. 该过程中不会创建新的活动点 $p=(x、\ y)$，其中 $x \neq j$。**2.** 如果 $\gamma_z < \alpha$，则该过程中删除的点是（非活跃）点 p_k，其中$k \in [w+1 \ldots r]$对于某个 $w < r < w'$成立。位于 p_w与 $p_{w'}$之间的其他点不会被LeftLinearWave(i,j,α)删除。**3.** 如果 $\gamma_z \geq \alpha$，则 q_{z+1} 要么被删除，要么变为非活跃。

证明。设 ℓ 是从 $p_w = q_z$ 出发的射线。假设该过程在找到 ℓ 下方的第一个点 $p^* =$（x^*，y^*）时终止（唯一不以这种方式结束的过程，是以到达（j，$A[j]$）而终止的过程）。射线发射过程最多添加两个斜率递减的新点 p' 和 p''，因此该过程不会创建新的活动点。p' 的斜率递减，是因为进入 p' 的线段是 ℓ（的一个子线段），而离开 p' 的线段通向 ℓ 下方的一个点。p'' 的斜率递减，是因为进入 p'' 的线段是一条从 p'（ℓ 上的一个点）出发的直线，而离开 p'' 的线段通向 ℓ 下方一条线段的后缀。

考虑 $\gamma_z < \alpha$ 的情况。此时，q_{z+1} 位于从 q_z 出发、斜率为 α 的射线下方。因此，射线发射过程在 p_{w+1} 之后、q_{z+1}之前的某个点处终止。由于不会创建新的活动点，下一条射线将从 q_{z+1} 或更靠后的位置发射，因此 LeftLinearWave$(i,\ j,\ \alpha)$ 不会删除 q_z 和 q_{z+1} 之间的其他点。

现在考虑这种情况 $\gamma_z > \alpha$。那么，q_z 和 q_{z+1} 之间的超线段位于从 q_z发射、斜率为 α 的射线之上。根据引理 15，q_z 和 q_{z+1}之间的所有（被动）点也都位于这条射线之上。因此，q_{z+1} 会被射线发射过程删除。

最后，考虑这种情况 $\gamma_z = \alpha$。那么，q_z 和 q_{z+1} 之间的超线段与从 q_z发射、斜率为 α 的射线重合。根据引理 15，q_z和 q_{z+1} 之间的所有（被动）点都会被射线发射过程删除。令 w' 满足 $q_{z+1} = p_{w'}$。如果$\alpha_{w'} \geq \alpha$，那么 q_{z+1} 会被该过程删除。否则，$\alpha_{w'} < \alpha$，因此射线发射过程在 q_{z+1}处终止。由于 q_z 和 q_{z+1} 之间的所有被动点都已删除，q_z和 q_{z+1} 在 P 中变为相邻点，对应线段的斜率为 $\gamma_z = \alpha$。但是，从 q_{z+1} 开始的线段的斜率为 $\alpha_{w'} < \alpha$，因此 q_{z+1} 变为被动点。◀

我们将具有 $\gamma_z > \alpha$ 长 射线的射线称为长射线，将具有 $\gamma_z \leq \alpha$ 短 射线的射线称为短射线。由于长射线会使 $\mathcal{P}_{\text{active}}$ 减小，我们可以像预热阶段那样显式处理它们：将过程中删除被动点的开销记在其创建上，并将过程结束时至多两个被动点的插入开销记在 $|\mathcal{P}_{\text{active}}|$ 的减少上。不使 $|\mathcal{P}_{\text{active}}|$ 减小的短射线将采用延迟方式处理。具体而言，不是在从活动点 q_z 开始的超线段中显式发射短射线，而是只存储该射线的斜率，并将其执行推迟到有需要时（例如由查找操作触发）。注意，在此超线段中随后发射的短射线可能会进一步改变所存储的斜率，后续的长射线也可能对其产生影响。下面将对此进行详细说明。

3.3 数据结构

由于我们的数据结构采用惰性策略，它维护的点序列将不同于$\mathcal{P}$ 若采用~第3.1节中的预热算法时所维护的点序列。因此，我们用$\mathcal{P}$ 表示~ ~数据结构实际维护的点集。这些点$\mathcal{P}$ 以通常方式定义线性线段 $\ell_i(x)$。对于 $x \in [1,n]$ ~ ~，我们用 $A[x]$ 表示值 $\ell_i(x)$，其中 ℓ_i 是包含 x的线段。需要强调的是，我们的算法并不维护$\mathcal{P}$。不过，仅为便于描述和分析，我们仍将继续提及原始的$\mathcal{P}$和数组 A。现在，活动点和被动点的定义、超线段的斜率 γ 以及短射线和长射线的定义，均以

各 $\tilde{\ell}_i$的斜率为准。[7] 不过，我们将保持如下性质：相对于$\mathcal{P}$ 和$\mathcal{P}_{\text{active}}$ 的活动点集合相同：

▶ **不变量 1**。$\mathcal{P}_{\text{active}} = \mathcal{P}_{\text{active}}$。

按照第 3.1 节的方法，我们在前驱/后继数据结构中维护 $\mathcal{P}$，并使用区间加数据结构 D_α 和 D_β 来表示线性线段 $\ell_i(x)$，这些线段由 $\mathcal{P}$ 中的点定义。通过以类似于第 AddConst、AddGradient 和长射线发射的方式实现这些操作（具体细节将在下文说明），我们将维护这一部分数据结构，使其始终满足不变量：它能够正确表示活动点的值。[8]

我们使用前驱/后继数据结构，根据活动点 $P\,q_2$（、、.....）的 q_z ∈坐标来维护活动点集合。对于每个 P，我们在区间加数据结构 中维护从 出发的超线段的斜率。此外，我们还通过在数据结构 中维护一条斜率为 、经过 的待处理短射线（见图 6），来维护它。这一数据结构称为加最小值数据结构，下文将对其作概述，并在第 4 节详细介绍。

▶**引理 17** (加最小值数据结构)。 存在一种数据结构，能够在 $O(\text{多对数} n)$ 时间内对点集 S 支持以下操作。
1. 插入*(x, y)* - 将点 (x, y) 插入到 S。 *2.* 删除*(x)* - 如果点 $p = (x, y)$ 存在，则将其从 S 中删除。 *3.* 查找 *(x)* - 返回满足 $p = (x, y)$ 属于 S 的 y，否则报告不存在这样的点。 *4. AddToRange(i, j, c)* - 对每个满足 $x \in [i \dots j]$ 的 $p = (x, y)$ 将 $y \leftarrow y + c$ 设为。 *5. Min(i, j, c)* - 对每个满足 $x \in [i \dots j]$ 的 $p = (x, y)$ 将 $y \leftarrow$ 设为 *min(y, c)*。

注意，存储 ρ_z 足以计算 $r_z(x)$，因为决定 r_z 的自由系数的活动点 q_z 已由 D_α 和 D_β 正确表示。我们将证明，只存储一条待发射射线就足以表示超线段中的所有待处理更改。此性质依赖于维护以下不变量。

▶ **不变量 2**。 对于每个活跃点 q_z，都有 $\rho_z > \gamma_z$。（回顾一下，γ_z 是连接 q_z 和 q_{z+1} 的超线段的斜率。）

其思想是，在这种表示下，对于任意 x，$A[x]$ 的值由以下两者中的最小值给出：包含 x 的 $\mathcal{P}$ 线段的值 $A[x] = \ell_w(x)$，以及包含 x 的超线段所对应的待处理短射线的值 $r_z(x)$。这由数据结构维护的以下主要不变量所体现。

▶ **不变式 3**。 设 $x \in [1, n]$，并设 p_w 和 q_z 分别为 x 在 P 和 P_{active} 中的前驱。则有 $A[x] = \min(\ell_w(x)\ r_z(x))$。此外，如果 $p = (x, A[x])$ 是 P 中的一个活跃点，则 $A[x] = A[x]$。

注意到，不变量 3 的第一部分与不变量 1 一起蕴含了不变量 3 的第二部分。这是因为，对于 P_{active} 中的活动点 $p = (x, A[x])$，x 的前驱就是它自身。由于 $\text{P}_{\text{active}} = \text{P}_{\text{active}}$，我们有 $p \in \text{P}_{\text{active}}$ 同样是 P_{active} 中 x 的前驱。根据定义，r_z 经过 $p = q_z$，因此 $r_z(x) = A[x]$，并且根据定义，$A[x] = \ell_w(x)$。因此，

[7] 使用 $\tilde{\alpha}$、$\tilde{\beta}$和 $\tilde{\gamma}$、会更准确，但这样会过于繁琐，因此我们仍使用 α、β、γ。 [8] 参见不变量 3 以及其后的注释。

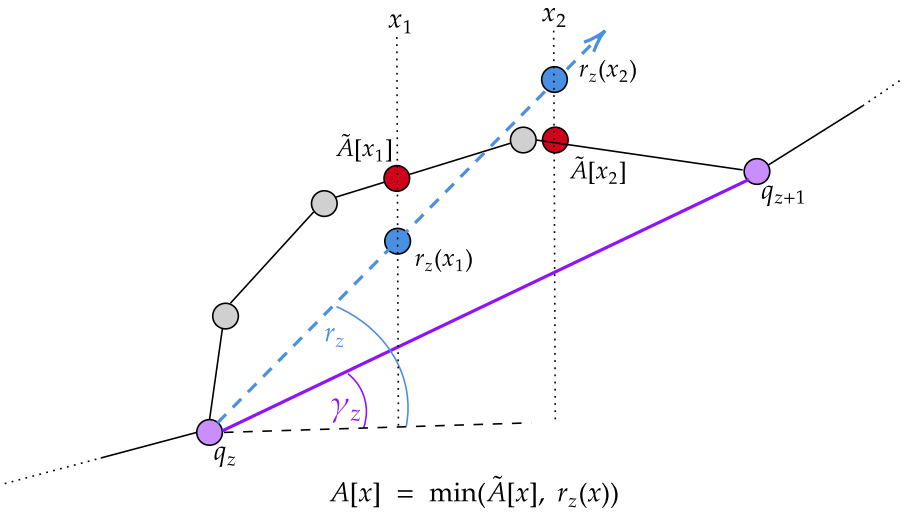

图6 两个连续活动点 q_z 和 q_{z+1}（紫色点）之间的超线段所存储数据的示意图。斜率 γ_z 是该超线段的斜率。存储在 q_z 中的斜率 $\rho_z > \gamma_z$ 表示一条待发射射线 r_z（蓝色虚线），它应从 q_z发出。$A[x]$ 的值是 $r_z(x)$（一个蓝色点）与 $\tilde{A}[x]$（一个红色点）之间的最小值；后者是由 $\tilde{\mathcal{P}}$ 定义的分段线性函数的值（灰色）。

在证明不变量得到保持时，我们无需明确证明不变量 3 的第二个陈述。

起初，$\mathcal{P} = \tilde{\mathcal{P}} = \{(1, 0), (|A|, 0)\}$，且 $\rho_1 = \rho_2 = \infty$。事实上，$A[x] = \min(A[x], r_1(x)) = \min(0, \infty) = 0$，且不变量 3 成立。接下来需要说明该数据结构所支持的各种操作如何实现，证明这些不变量得以维持，并分析运行时间。

刷新操作。我们首先介绍一种服务操作 flush(q_z)，它会显式发射从活动点 q_z 出发、位于超线段中的待处理短射线。在执行查找操作之前调用 flush 会很有用；执行其他操作时也可以调用它，以确保惰性实现在预热阶段能够正确跟随显式实现。这对于可能创建 $O(1)$ 个新活动点、从而改变超线段划分的操作尤为重要，同时也有助于简化正确性证明。请回想，我们最初之所以避免发射局部射线，是因为这类射线可能很多，而且我们无法承担在每条射线末端创建 $O(1)$ 个新被动点的代价。不过，我们可以在每次 Lookup、AddConst 或 AddGradient 操作之前执行 $O(1)$ 次刷新操作，因为添加这 $O(1)$ 个新点的代价可以计入操作本身。

对 $q_z = p_w \in \mathrm{P}_{\text{active}}$ 执行刷新，步骤如下。从 p_{w+1} 开始，扫描 P 中的点。扫描到 $p = (x, y)$ 时，比较 y 和 r_z。如果 $r_z(x) \le y$，就从 P 中删除 p。否则，扫描停止。令 p_{end} 为扫描停止时所在的点。如果扫描过程中没有删除任何点，则设置 $\rho_z = \infty$ 并终止。否则，令 p_{del} 为扫描删除的最后一个点。我们计算 $p^* = (x^*, y^*)$，即 r_z 与经过 p_{del} 和 p_{end}的直线 ℓ 的交点。最后，向 P 中插入 $p' = (\lfloor x^* \rfloor, r_z(\lfloor x^* \rfloor))$ 和 $p'' = (\lceil x^* \rceil, \ell(\lceil x^* \rceil))$（如第 3.1 节的预热算法所示），用在 p' 或 p'' 处结束和开始的线段的新参数更新 D_α 和 D_β，并设置 $\rho_z = \infty$。

▶**引理 18.** 对一个活动点应用刷新 $q_z \in \mathcal{P}_{\text{active}}$ 可保持不变量 1–3。~此外，这还能保证 $\mathcal{P}$ 和 $\mathcal{P}$ 在（非活跃）点 q_z~和 q_{z+1} 之间的限制相同，并且对每个 $x \in [x_z \dots x_{z+1}]$, $A[x] = A[x]$。

证明。不变量 2 得以保持，因为刷新操作将 ρ_z 设为 ∞。由于 $\rho_z > \gamma_z$，引理 16 保证刷新操作的扫描在 q_{z+1} 或之前结束。因此，不变量 1 得以保持，因为 $\mathrm{P}_{\text{active}}$ 不变，且刷新操作只会删除 P 的被动点~。接下来证明不变量 3 得以保持。注意，刷新操作结束时 ρ_z 被设为 ∞，且 q_z 仍然是每个 $x \in [x_z \dots x_{z+1}]$ 的前驱活动点，所以~我们需要证明 $A[x] = A[x]$。令 $x \in [x_z \dots x_{z+1}]$。若 $x \le x^*$，那么在应用刷新操作之前，~~我们有 $A[x] \ge r_z(x)$，因此根据不变量 3 $A[x] =$，$\min(A[x], r_z(x)) = r_z(x)$。由于~刷新操作将 $A[x]$ 的值设为 $r_z(x)$（对于 $x < x^*$），不变量 3 仍然成立。若 $x > x^*$，则刷新操作不会改变 $A[x]$ 的值。由于连接 p_{del} 和 p_{end} 的直线 ℓ 起点不低于 r_z，终点则低于 r_z，其斜率小于 ρ_z。由于 p_{end} 和 q_z 之间的点（不包括 q_z）均为被动点，相应线段的斜率也低于 ρ_z~，因此对于每个 $x \in$，$(x, A[x])$ 都低于 r_z $(x' \dots x_{z+1}]$。根据应用刷新操作之前的不变量 3，~~我们有 $A[x] = \min(A[x], r_z(x)) = A[x]$。因此，赋值 $\rho_z \leftarrow \infty$~且不改变 $A[x]$，即可满足不变量 3。◀

3.4 实现数据结构

~查找(k)。要执行查找(k) 我们先取出前驱 q_z，它是 k 在 $\mathcal{P}_{\text{active}}$中的前驱，然后调用~~刷新$(q_z)$。随后，我们取出 p_w 在 k 于 $\mathcal{P}$ 中的前驱，并返回 $\ell_w(k)$，根据引理 18，该结果正确。此操作显然能维持全部三个不变量。

AddConst(i, j, c)。与预热算法（第 3.1 节）中的实现类似，我们首先执行查找查询，以获取 $x \in \{i-1, i, j, j+1\} =$ B 对应的 $A[x]$。设 $\mathrm{P}_{\mathcal{B}}$~为得到的 $O(1)$ 个点。我们假设 $\mathrm{P}_{\mathcal{B}}$ 中的点此前均不在 P~中（其他情况的处理方式类似）。我们将 $\mathrm{P}_{\mathcal{B}}$ 中的点插入 P，并使用 $O(1)$ 次操作相应地更新D_β 和 D_α。注意，此时所有包含 $\mathrm{P}_{\mathcal{B}}$ 中点的 $O(1)$ 个大型线段都已刷新（这是因为调用了查找）。接下来，我们应用D_β.AddToRange(i, j, c)。注意，这会改变 $i-1$ 与 i 之间以及 j 与 $j+1$之间的线性线段。我们使用额外的 $O(1)$ 次操作更新 D_α 和 D_β，以反映这些变化。

~接下来，对于每个 $p_k = (x_k, y_k) \in \mathcal{P}_{\mathcal{B}}$，如果 p_k 刚变为活跃，则将 x_k 插入 $\mathcal{P}_{\text{active}}$，并将 ρ 在 x_k 中的值设为 ∞。否则，如果 p_k 刚变为非活跃，则从 $\mathcal{P}_{\text{active}}$ 中~删除 x_k，并在必要时 D_γ。最后，如果 $\alpha_k = \alpha_{k-1}$，则通过从 $\mathcal{P}$ D_α 中删除 p_k 并 D_β，合并这两条线段~。

▶ **引理 19.** 应用 AddConst(i, j, c) 保持不变量 1–3。

证明。 由于 $\mathcal{P}$ 中因 AddConst(i, j, c) 而变为活跃或非活跃的点只有 $\mathcal{P}_{\mathcal{B}}$中的点。由于包含 $\mathcal{P}_{\mathcal{B}}$ 中点的超级线段已被刷新，引理 18 以及 AddConst 对 $\mathcal{P}_{\mathcal{B}}$ 中变为活跃或非活跃的点进行的显式处理，保证了不变量1 得以保持。

同理，只有包含 γ 中因 AddConst(i, j, c) 而改变其值的超级线段，才是那些包含 $\mathcal{P}_{\mathcal{B}}$中点的超级线段。这些超级线段的值 ρ 均通过刷新或显式操作设为 ∞。因此，不变量2 成立。

为证明不变量 3，令 $k \in [1 \dots |A|]$，并令 q_z为应用 AddConst$(i,\ j,\ c)$ 之前的活跃点，且满足 $k \in [x_z \dots x_{z+1}]$。

- 如果 $[x_z \dots x_{z+1}] \cap [i-1 \dots j+1] = \emptyset$，那么更新不会影响 $A[k]$ 和 $A[k]$。此外，更新后，k 的前驱活动点仍为 q_z，并且更新不会影响 ρ_z。因此，$\min(A[k], r_z(k))$ 不会改变。

- 若 $[x_z \dots x_{z+1}] \subseteq [i+1 \dots j-1]$，则注意到，$q_z$ 在更新后仍然活跃，因为 α_z 和 α_{z-1} 不受该更新影响。$A[k]$ 的值以及 q_z 的 y 坐标通过 D_β.AddToRange(i, j, c) 操作增加了 c。ρ_z 的值没有改变，因此 $r_z(k)$ 也增加了 c。由此可得，$\min(A[k], r_z(k))$ 增加了 c，符合要求。

- 若 $[x_z \dots x_{z+1}] \cap B \neq \emptyset$，则注意，我们对 q_z 执行了刷新操作，因此在应用 D_β.AddToRange(i, j, c) 之前，有 $\rho_z = \infty$ 和 $A[k] = A[k]$。于是，刷新操作之后，有 $A[k] = \min(A[k], r_z(k)) = \min(A[k], \infty) = A[k]$。应用更新 AddConst$(i, j, c)$ 后，k 的前驱活动点要么是 q_z，要么是 P_B 中的某个点（如果操作使 P_B 中的某个点变为活跃），要么是 P_B 中某个点的前驱活动点（如果 P_B 中的某个点 q_z，并因该更新而变为非活跃）。在所有这些情况下，更新后的表示中，k 的前驱活动点 q_a 具有 $\rho_a = \infty$（因为它要么是一个新的活动点，要么是一个已存在且执行过刷新的活跃点）。$A[k]$ 的值通过操作 D_β.AddToRange(i, j, c) 增加了 c。综上，有 $\min(A[k], r_a(k)) = \min(A[k], \infty) = A[k]$ 由于 $A[k]$ 在必要时增加了 c，因此它现在表示 AddConst(i, j, c) 操作之后 $A[k]$的值。◀

AddGradient(i, j, c)。与预热算法中的情况一样，AddGradient 的实现与 AddConst 类似，不同之处在于，它不是对 D_β 应用 AddToRange(i, j, c)，而是对 D_α 应用该操作，从而将 i 和 j 之间所有线段的斜率增加 c。同样，我们使用 O 对 D_γ 调用 AddToRange c 次，以增加相应超级线段的斜率（包含 i 和 j 的超级线段需要特殊处理，因为它们的斜率增加量可能小于 c）。最后，我们对 D_ρ 使用 AddToRange，以增加待处理射线的斜率。

▶ **引理 20**。应用 AddGradient(i, j, c) 保持不变量 *1–3*。

证明。不变量1 的证明与引理 19中的证明完全相同。不变量 2 成立，因为只有那些包含以下点的超级线段，其 γ 和 ρ 才会以不同的值发生变化：$\mathcal{P}_B$；这些超级线段会被刷新，并由实现显式处理。

对于不变量 3，令 $k \in [1 \dots |A|]$ 并令 q_z 为满足以下条件的活动点：在应用 $k \in [x_z \dots x_{z+1}]$之前 AddGradient$(i, j, c)$。

- 若 $[x_z \dots x_{z+1}] \cap [i-1 \dots j+1] = \emptyset$，则不会发生任何变化，就像引理 19 的证明中一样。若 $[x_z \dots x_{z+1}] \subseteq [i+1 \dots j-1]$，注意到 q_z 仍然是活跃的，因为 α_z 和 α_{z-1} 都通过 c增加了。$A[k]$ 的值通过 D_α.AddToRange(i, j, c) 操作增加了 $k \cdot c$。q_z 的 y 坐标通过同一操作增加了 $x_z \dots c$。ρ_z 的值也增加了 c，因此 $r_z(k)$ 增加了 $x_z + (k - x_z) \cdot c = k \cdot c$。由此可知，$\min(A[k], r_z)$ 增加了 $k \cdot c$，符合要求。若 $[x_z \dots x_{z+1}] \cap B \neq \emptyset$，则 $A[k] = A[k]$ 可根据引理 19 中相应情形证明的相同论证得出。◀

LeftLinearWave(i, j, c)。在对算法的描述中，我们会说它从某个点 p显式地执行射线发射过程。其含义如下。首先，刷新包含 p 的超线段。然后，我们从 p 开始扫描 $\mathcal{P}$ 中的点，使用

˜P 中的后继查询。与第 3.1 节的预热算法类似，如果射线 ℓ （斜率为 c，从p发出）满足 $y_k \geq \ell(x)$，我们就从 P 中删除˜扫描点 $p_k = (x_k, y_k)$。如果扫描到达活动点 $q_w = (x_w, y_w)$，并发现 q_w 不在 ℓ 之下——我们会在从 P 中删除 x_w 之前，对 q_w 执行刷新操作˜。我们还会从 P˜ ˜ ˜中删除 $x_{w\ \text{active}}, D_\gamma, D_\rho$，并相应地更新 P 和 $\mathrm{P}_{\text{active}}$ 中 q_w 的前驱的斜率。˜到达位于射线 ℓ 下方的点 p_k （或到达 p_b）时，我们按第 3.1 节所述将点 p' 和 p'' 加入 P（如有必要，删除 p_b 并加入 $(j, \ell(j))$），射线发射过程随即终止。

现在我们描述 LeftLinearWave$(i,\ j,\ c)$ 的算法。（参见图 7）。首先，与˜ 中的 AddConst 和 AddGradient 操作一样，我们将 $\mathcal{P}_\mathcal{B}$ 中的点加入 $\mathcal{P}$，并刷新每个包含 $\mathcal{P}_\mathcal{B}$ 中点的超线段。如果 $\alpha_a > c$，我们就从 p_a 显式执行射线发射过程。令 p 为显式射线发射过程终止的位置，或者$p = p_a$ 如果 $\alpha_a \leq c$。

令 $q_w = (x_w,\ y_w)$ 为 p 之后的第一个（弱）活动点。我们使用 D_γ.nextGT$(x_w,\ c)$，得到下一个长射线发射应从其开始的点 $q_z = (x_z,\ y_z)$。接着，我们在每个从满足 $x_t \in [x_w \ldots x_z)$ 的活动点 $q_t = (x_t,\ y_t)$ 开始的超线段中隐式发射短射线。这通过在维护 ρ_z 值的加最小值数据结构 D_ρ 上执行一次 Min$(x_w,\ x_{z-1},\ c)$ 操作来完成。此操作的效果是将所有此类 t' 的值设为$\rho_t \leftarrow \min(\rho_t,\ c)$。接下来，我们从 q_z 显式执行长射线发射。

我们不断重复上述段落，并让上一次显式长射线发射过程终止的位置承担 p的角色。如果一次显式射线发射到达j，或者 D_γ.nextGT$(x_w,\ c)$ 返回 'null' 或一个位于 j之后的点，我们就停止。在后一种情况下，令 $q_{b'}$ 为包含 j的超线段的起点。我们隐式发射所有超线段中的短射线，这些超线段的起点不早于 p，终点不晚于 $q_{b'}$。最后，我们调用 flush$(q_{b'})$；如果 $\alpha_{b'} > c$，则从 $q_{b'}$显式执行射线发射过程。

最后，我们还需要更新 i 和 j周围的效应。对于每个 $p_k \in \mathcal{P}_\mathcal{B}$，我们检查 p_k 是否活跃，并据此更新 D_γ （类似于 AddConst 中的这项更新）。

▶ **引理 21**。 应用 LeftLinearWave$(i,\ j,\ c)$ 可保持不变量 *1–3*。

证明。 我们先证明以下断言。

▷ 断言 22. LeftLinearWave 的实现执行的长射线发射过程与第 3.1 节的预热算法完全相同（起点、终点以及删除的点都相同）。

证明。 由于包含 p_a 的超级线段已被刷新，我们的算法会从p_a 发射射线，当且仅当预热算法也会这样做。如果我们没有从 p_a发射射线，我们的˜算法就会从第一个满足条件的活动点发射一条长射线 q_z，其中 $\gamma_z > c$。由于 $\mathcal{P}_{\text{active}} = \mathcal{P}_{\text{active}}$（不变量 1），存储在 γ 中的斜率 D_γ 是$\mathcal{P}$的超级线段的斜率。因此，预热算法发射的第一条长射线不会从 q_z之前的位置起始。预热算法可能从满足条件的活动点 q_w发射了若干次短射线 $w < z$，但根据引理 16，这些短射线的影响仅限于起始于 q_w的超级线段，不会影响任何活动点。因此可以保证，预热算法发射的第一条长射线也从 q_z发射。

考虑从 q_z （或从 p_a，取决于α_a）开始执行的第一次长射线发射过程。在整个扫描过程中，我们总是在扫描超线段中的点之前，对超线段执行刷新操作。因此，根据引理 18，可以保证我们看到的点与预热算法扫描时看到的点完全相同。由此可知，该扫描过程会创建出与预热算法完全相同的点后终止，并删除所有

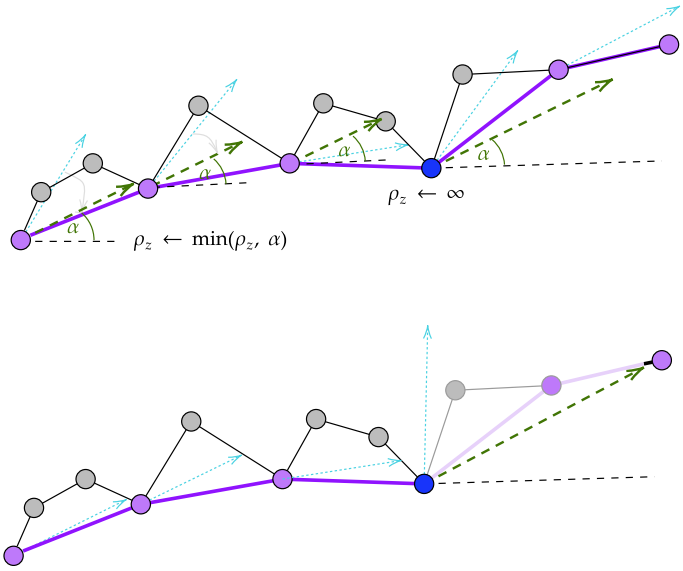

图 7 首次显式长射线发射过程的示意。首先，确定最左侧的活跃点 q_z，其 γ_z 大于 α（蓝色），并假设无需从 p_a 进行射线发射过程。然后，将 q_z 之前的活跃点的 ρ 值（浅蓝色射线）赋值为 $\rho \leftarrow \min(\rho, \alpha)$。最后，从 q_z 显式执行射线发射过程。由于显式射线发射过程首先会刷新 q_z 所属的超线段，因此 ρ_z 的值被设为 ∞。

位于这两个点之间的点，并 q_z）。然后，我们的算法使用 D_γ.下一个大于点来确定下一条长射线应从何处开始。对所有后续长射线重复上述推理，即可通过归纳得到该结论。◀

不变量 1 根据声明 22，每个因射线$_{\text{active}}$ 发射过程而从 $\tilde{\mathcal{P}}$中删除的活跃点，也会从 $\mathcal{P}_{\text{active}}$ 中删除（且只有这些点）。根据引理16，射线发射可能导致变为活跃的唯一点是 $p = (j, A[j]) \in P_{\mathcal{B}}$。算法通过刷新包含 j 的超线段来显式处理这一情况，从而保证在 p 处结束和开始的线段在 $\mathcal{P}$ 和 $\tilde{\mathcal{P}}$ 中的斜率相同。因此，p 在 $\mathcal{P}$ 中活跃，当且仅当它在 $\mathcal{P}$中活跃，不变量 1 得以满足。

不变量 2 设 q_z 是在应用 LeftLinearWave(i, j, c) 后 $\mathrm{P}_{\text{active}}$ 中的一个活动点。注意，由于射线发射不会创建任何新的活动点（可能出现点 $(j, A[j])$ 的情况除外），因此点 q_z 在应用 LeftLinearWave(i, j, c) 之前也处于活跃状态（或者 q_z 是 P_B 中的一个点）。如果调用了 flush(q_z)，则 ρ_z 会被设为 ∞，不变量 2 得以满足。

否则，必定是因为未从q_z执行长射线发射过程。此外，没有任何长射线发射删除超线段 q_z中的点。因此可知，q_{z+1} 也未受到更新的影响，因为从 q_w 开始且带有$w \geq z+1$ 的射线发射不会改变 q_{z+1}的值，而可能从 q_z 执行的短射线发射过程同样不会改变 q_{z+1}。因此，LeftLinearWave(i, j, c)不会改变 γ_z。由于没有长射线从 q_z开始，且 q_z未被长射线发射删除，我们必有 $\gamma_z < c$。在应用LeftLinearWave之前，我们有 $\gamma_z < \rho_z$。ρ_z 要么在算法中保持不变，要么被短射线设为min(c, ρ_z)。在这两种情况下，不变量 2均得到保持。

不变量 3 设 $x \in [n]$。关于包含$[x_z \dots x_{z+1}]$ 的超线段 x 在应用LeftLinearWave(i、 j、 c)之前的情况，我们分为两种情形。

- 若 q_z 被显式长射线发射删除。令 q_w 为长射线的起点。因此，应用 LeftLinearWave(i, j, c) 后， q_w 是 x的前驱活动点，并且从 q_w 开始（且包含 x）的超线段会被刷新。因此，算法设置 $\rho_w = \infty$，并在该线段上应用与预热算法 $A[x] = A[x] = \min(A[x],\ r_w(x))$ 完全相同的˜ ˜变更，正如所要求的那样。

- 若 q_z 未被显式长射线发射删除。由断言 22 以及预热算法的正确性可知，如果需要修改 $A[x]$ 的值，那么这是短射线发射的结果。如果预热算法没有从 q_z执行射线发射过程，那么必有 $\gamma_z < c$。在这种情况下，˜赋值 $\rho_z \leftarrow \min(\rho_z, c)$ 不会改变 ρ_z，因此 $\min(A[x], r_z)$ 不会改变，正如所要求的那样。接下来处理预热算法从 q_z 发射斜率为 c 的局部射线 r 的情况。令 $A_{\mathsf{before}}[x]$ 和 $A_{\mathsf{after}}[x]$ 分别为应用 LeftLinearWave(i, j, c) 前后 $A[x]$的值。 $A_{\mathsf{after}}[x]$ 的值为 $A_{\mathsf{after}}[x] = \min(A_{\mathsf{before}}[x], r(x))$。令 r_z^{before}（分别为 r_z^{after}）表示应用 LeftLinearWave(i, j, c) 前（分别为后）位于 q_z、斜率为 ρ_z（分别为 $\min(\rho_z, c)$ ）的待发射射线。注意，$r_z^{\mathsf{after}}(x) = \min(r_z^{\mathsf{before}}(x)\ r(x))$，因此

$$\min(\tilde{A}[x], r_z^{\mathsf{after}}(x)) = \min(\tilde{A}[x], r_z^{\mathsf{before}}(x), r(x)) =$$
$$\min(\min(\tilde{A}[x], r_z^{\mathsf{before}}(x)), r(x)) = \min(A_{\mathsf{before}}[x], r(x)) = A_{\mathsf{after}}[x]$$

如所要求。 ◀

˜**复杂度**。 我们首先证明，在一系列 s 操作中，加入到 $\mathcal{P}_{\mathsf{active}}$ 的活动点数量为 $O(s)$。这是因为刷新操作不会向 $\mathcal{P}_{\mathsf{active}}$添加活动˜点，而每次调用 AddConst、AddGradient 或 LeftLinearWave 至多会创建O(1) 个活动点。

考虑一系列 s操作。我们使用标准的摊还分析证明，˜每次操作的摊还时间为 O(1)。唯一的难点在于如何摊还显式执行的射线发射所花费的时间：这些射线发射是在任何一次 flush 调用中以及执行 LeftLinearWave 时进行的；此外，还需摊还执行 LeftLinearWave 时隐式进行的短射线发射所花费的时间。

˜我们将处理包含 $\mathcal{P}_{\mathcal{B}}$ 中各点的超级线段所需的 O(1) 时间摊给每次操作，其中包括向 $\mathcal{P}_{\mathcal{B}}$ 插入 O(1) 个新点所需的时间，但不包括对这些超级线段调用˜flush 的时间。同样，我们将 AddConst 和 AddGradient 中的 O(1) 时间更新 D_α 和 D_β 摊给相应操作本身。

˜ ˜每次调用刷新都可能从$\mathcal{P}$ 中删除许多被动点，此外，还需要花费 O(1)˜ ˜时间将 O(1)个新的被动点插入$\mathcal{P}$。删除$\mathcal{P}$ 中每个点的时间 p 都计入 p的插入操作。在包含$\mathcal{P}_{\mathcal{B}}$中点的超级线段上调用刷新，或˜在包含 LeftLinearWave 中点 p_a 的超级线段上调用刷新时，所需的额外 O(1)时间计入发起该调用的操作。所有其他刷新调用都发生在 LeftLinearWave 中的显式长射线发射期间，相关开销将在下一步计入。

在一次显式长射线发射过程中遇到的每个超线段 $[q_z, q_{z+1}]$（最后一个除外），都会导致从 $\mathsf{P}_{\mathsf{active}}$ 中删除活动点 q_{z+1} 。对于每个这样的˜超线段，我们将检查 q_z 并调用 flush(q_z) 所花费的 O(1) 时间，计入从 $\mathsf{P}_{\mathsf{active}}$ 中删除 q_{z+1} ˜ 的开销。对于最后一个超线段，根据引理 13， q_{z+1} 要么被删除，要么˜ ˜变为非活跃，因此我们将 $\mathsf{P}_{\mathsf{active}}|$ 减少 1 所花费的 O(1) 时间计入 |。注意，处理˜最后这条线段时，可能会向 P 中插入 O(1) 个新的被动点，这在˜已计入的 O(1) 时间预算之内。

最后，我们将以隐式方式实现短射线发射时调用 D_ρ.Min 所花费的时间计入开销。这些调用均由某个 LeftLinearWave 操作产生。我们将 $O(1)$ 次此类调用计入 LeftLinearWave 操作本身，其余调用则分别计入此前的长射线发射。

~ ~每次操作以及 $|\mathcal{P}_{\text{active}}|$ 的每次减少均计费为 $O(1)$ 时间。由于该序列包含 $O(s)$ 次操作，并且正如我们所证明的，曾变为活跃状态的点有 $O(s)$ 个，因此处理整个序列的~总时间为 $O(s)$。

处理 RightLinearWave。 如上所述，处理RightLinearWave 与 LeftLinearWave 是对称的，只需让算法从右向左执行，从 p_b 开始，并从活动点发射负射线（活动点的定义保持不变）。为了跟踪待处理的短负射线，我们维护一个额外的 Add-min 数据结构 ~ D'_ρ，现在 $A[x]$ 可表示为 $\min(A[x], r_z(x)\ r'_{z+1}(x))$，其中 r'_{z+1} 是经过 q_{z+1} 的待处理负射线。正确性证明、不变量的维护以及复杂度分析，都与 LeftLinearWave 的完全对称。至此，定理 4 的证明完成。

4 加最小值数据结构

本节介绍引理 17 中的加最小值数据结构。为解释该数据结构的主要思想，我们假设没有点被添加或删除，并专注于仅支持 Min(i, j, c) 和 AddToRange(i, j, c)。我们考虑按 x 坐标排序的点 $p_1 =(x_1, y_1)$ $p_2 = (x_2, y_2)$ .. 。加最小值数据结构是递归的。它由一个区间加法数据结构 D 和一个递归的加最小值实例 R 组成，后者仅存储点的一个常数比例（2/3）。

这些点被划分为连续点对。点 $p = (x, y)$ 的代表点是该点所属点对中的第一个点。令 $M(x)$ 表示 p 的代表点的 x 坐标。最初， D 存储所有点 $p_k = (x_k, y_k)$，而 R 仅存储代表点，这些代表点被初始化为 ∞。即，(x_1, ∞)、(x_3, ∞)、(x_5, ∞) .. 。我们维护不变量： $y_k = \min(D$.Lookup(x_k) R.Lookup$(M(x))$，因此，对该数据结构执行 Lookup(x_k) 查询时，可以通过在 D 上执行一次 Lookup，以及在 R 上递归执行一次 Lookup 来完成，总耗时为多对数时间。

我们使用服务操作 Assign(x_k, c)，当存在点 $p =(x_k, y_k) \in S$ 时，它会将 $y_k \leftarrow c$ 赋值为相应的值。操作 Assign(x_k, c) 的实现方式是令 D 存储值 c，如下所示（注意，使用 Remove 和 Insert 对 D 执行 Assign 很简单）。设 $p_{k'}$ 为与 p_k 成对的另一个点。我们通过一次 Lookup 操作获取 $y_{k'}$ 的值，并调用 D.Assign(x_k, c)、 D.Assign$(x_{k'}, y_{k'})$，然后递归调用 R.Assign$(M(x_k)\ \infty)$。

为了实现 Min(i, j, c)，令 p_a（分别为 p_b）表示第一个满足 $x_a \geq i$（分别为 $x_b \leq j$）的点。首先假设 p_a 是代表点（即 $M(x_a) = x_a$），而 p_b 不是，因此受影响范围 $[i, j]$ 恰好对应于一段连续点对范围。我们只需调用 R.Min(i, j, c)，显然即可正确实现 Min 操作，同时维持不变量。如果 p_a 不是代表点，则需要以不同方式处理包含 p_a 的点对，因为我们不希望影响 p_{a-1} 的值。我们通过两次 Lookup 操作在更新前获取 y_{a-1} 和 y_a 的值，并调用 D.Assign(x_{a-1}, y_{a-1})、 D.Assign$(x_a, \min(y_a, c))$、 R.Assign(x_{a-1}, ∞) 来赋值（最后一次调用是递归赋值）。这样既能维持不变量，也能确保 p_{a-1} 和 p_a 都得到正确表示。对于 p_b 是代表点的情况，处理方法类似。

我们以类似的思路实现 AddConst(i, j, c)。如果 p_a 是代表点而 p_b 不是，我们只需调用 D. AddToRange(i, j, c) 和 R.AddToRange(i, j, c)。否则，我们将按照 Min 中所述的方式显式处理区间的

按照为 Min 描述的方式显式表示区间的端点。

为了支持点的插入和删除，我们不能再使用刚性的连续点对划分。相反，我们将采用标准技术，将点划分为由单个点或两个连续点组成的线段。每个线段中只有第一个点会在递归结构中表示。我们确保在两个连续线段都只包含一个点时将它们合并。这保证了线段数至多为 $2/3$ 点数，因此递归结构足够小。

为了跟踪线段划分，我们在前驱数据结构 M中维护各代表点。点 p_k 的代表点即为 x_k 在 M中的前驱。不变量现在变为 $y_k = \min(D.\text{查找}(x_k)\ R.\text{查找}(M.\text{前驱}(x_k))$. 若存在坐标为 D （分别为 R）的点，我们将该点的 y 坐标值记为 $D[x]$ （分别为 $R[x]$）。我们将 $M.\text{前驱}(x)$的值记为 $M[x]$ 。采用这些记号，我们维护的不变量为：

▶ **不变式 4.** 对于数据结构中的每个 $p = (x, y)$，都有 $y = \min(D[x], R[M[x]])$.

我们用 $s_1, s_2 \dots s_r$ 表示这些线段。关于这些线段的不变量为：

▶ **不变式 5.** 对于每个 $i \in [1 \dots r-1]$， s_i 的长度为二，或者 s_{i+1} 的长度为二。

请注意，不变式 5 的一个直接推论是 $r \le \lceil \frac{2n}{3} \rceil$。另外请注意，对于每个$i \in [1 \dots |D|]$, $M[x_i]$要么是 x_i，要么是 x_{i-1}。

为完整起见，我们给出该数据结构的所有细节。初始化时，数据结构中不包含任何元素，并且 $D, M,$ 和 R 均为空。

查找(x)。我们在 D 上执行查找(x) 查询，并在R 上执行（递归）查找$(M[x])$ 查询，然后返回两者中的最小值。

赋值$(x,\ y)$。设 $x_{i'} = M[x_i]$，并设 s_a为包含点 p 的线段，其 x 值为 $x_$。我们赋值 $D[x] \leftarrow y_$。若 s_a 中还有另一个点 $p' = (x',\ y')$，我们也赋值 $D[x'] \leftarrow y'$。注意，我们可以使用前驱和后继查询来确定哪两个候选点可能是 p'，并使用 M 确认其中哪一个位于 s_a 中。最后，我们（递归地）赋值 $R[x_{i'}] \leftarrow \infty$。

移位$(x,\ x')$。我们引入一种服务操作，它有助于在插入和删除过程中维护不变量。Shift 的输入是 D 中某个点 $p_i = (x,\ y_i)$ 的 x 值，以及一个满足 $x_{i-1} \le x' \le x_{i+1}$ 的新值 x' 。该操作 Shift 将 p_i 替换为 $p = (x',\ y_i)$。注意，由于对 x' 的约束，新点 p 可以进入移除 p_i 的线段。我们按如下方式实现 Shift$(x,\ x')$。首先，从 D 中删除 p_i，并插入 $p = (x',\ y_i)$ 来替代（需要执行查找操作以获取 y_i）。如果 $M[x_i] = x_i$，我们还从 M 中删除 x_i，并添加 x' 来替代。最后，如果 x_i 在 M 中被替换，我们还会递归地应用 $R.\text{Shift}(x,\ x')$。

插入$(x,\ y)$。设 x_p 和 x_s 分别为数据结构中 x 的前驱值和后继值。设 $x_a = M[x_p]$和 $x_b = M[x_s]$。

- 如果 $x_a \neq x_b$ 我们应用 $D.\text{插入}(x, y)$。接着，我们按如下方式更新 M 和 R：如果包含 x_s 和 x_p 的线段长度都为二，那么我们创建一个新的线段 $s = [(x, y)]$，并（递归地）应用 $R.\text{插入}(x, \infty)$。我们还将 x 添加到 M。如果包含 x_p 和 x_s 的线段中有一条长度为一，假设包含 x_p 的线段 s 长度为一。我们通过应用 $D[x_p] \leftarrow \text{查找}(x_p)$，将 $D[x_p]$ 的值更新为数据结构中 x_p 的正确值。此外，

我们应用 $R[x_a] \leftarrow \infty$ 以确保不变式 4。注意，在此操作中，我们向一条线段添加一个点。如果在此过程中，添加的点（x，y）成为此前只包含一个点 $p' =$（x'，y'）的线段中的第一个点，我们需要更新 M，使得 x' 映射到 x，并更新 R，使其包含一个 x 坐标为 x' 而非 x 的点。具体做法是在 M 中用 x' 替换 x，并应用 R.移位(x，x')。

- 若 $x_a = x_b$，则在包含 x_p 的线段 s 正前方插入点 (x', y')，属于我们已处理过的情况。因此，我们不将 (x, y) 插入 s，而是将 x_p 替换为 (x, y)，并按照前述情况在 s 前插入 x_p。设 $p_p = (x_p, y_p)$（通过 Lookup(x_p) 获得）。我们从 D 中删除值为 x_p 的点 x，并插入 (x, y)，以此将 p_p 替换为 $p = (x, y)$。我们还从 M 中删除 x_p，并改为添加 x。回想一下，$p_s = (x_s, y_s)$ 也在线段中，并且我们将 Lookup(x_s) 赋值给 $D[x_s]$，将 ∞ 赋值给 $\mathbb{R}[x_a]$。我们还应用 R.Shift(x_p, x)。这样，我们就将 p_p 替换为 $p = (x, y)$。接下来，我们通过前述情况之一插入 (x_p, y_p)。

删除(x)。令 $x' = M[x]$。令 s_i 为包含 x 的线段。

- 如果 s_{i-1} 和 s_{i+1} 的长度均为 2，或二者不存在，则通过应用 D.删除(x)，从 D 中删除 x。如果 s_i 的长度为 2，则可能出现这样的情况：s_i 中的第一个点已被删除，现在为 $p' = (x',\ y')$。如果确实如此，则在 M 中用 x' 替换 x，并应用 R.移位(x，x')。如果 s_i 的长度为 1，则应删除线段 s。我们通过应用 R.删除(x') 来删除 $R[x']$。我们还从 M 中删除 x'。

- 如果 s_{i-1} 或 s_{i+1} 的长度为 1（假设 s_{i-1} 的长度为 1）。注意，根据不变式 5 s_i 的长度为 2，因此 $s_{i-1} \cup s_i$ 恰好有 3 个点。令 p' 为 s_{i-1} 中的点，令 p'' 为 s_i 中除 (x, y) 之外的另一个点。我们按照前面各情况中的描述，从 s_{i-1} 中删除点 $p' = (x', y')$。然后通过对 D 执行操作，以及对 R 执行赋值和移位操作（如插入中所述），修改 s_i 中各点的 x 和 y 值，将它们替换为 p' 和 p''。注意，我们已经描述了如何删除长度为 1 的线段中的元素（由于不变式 5，如果 s_{i-2} 存在，则它必定长度为 2）。如果删除导致 s_i 中的第一个点发生变化，则按照插入中的方式相应地更新 M 和 R。

AddConst(i, j, c)。设 $p_p = (x_p, y_p)$ 和 $p_s = (x_s, y_s)$ 分别是数据结构中 i 的 x 后继和 j 的 x 前驱。设 $x_a = M[x_p]$ 和 $x_b = M[x_s]$。设 $s_{a'}$ 和 $s_{b'}$ 分别是包含 x_p 和 x_s 的线段。首先，我们更新 $s_{a'}$ 和 $s_{b'}$ 中所有需要改变其值的元素（最多 4 个）。对于 $s_{a'}$ 或 $s_{b'}$ 中满足 $x \in [i \ldots j]$ 的每个点 $p = (x, y)$，通过 Assign(x, Lookup(x) $+ c$) 操作完成此更新。设 $p_1 = (x_1, y_1)$ 是 $s_{a'+1}$ 中的第一个点，$p_2 = (x_2, y_2)$ 是 $s_{b'-1}$ 中的最后一个点；接下来只需将 c 加到范围 $[x_1 \ldots x_2]$ 内的所有元素上。由于每个点 (x, y) 的 y 值表示为 min($D[x]$, $R[M[x]]$)，我们将 c 加到该表示的两部分，具体如下。对于 D 中满足 $x \in [x_1 \ldots x_2]$ 的每个点 $p = (x, y)$，将 c 加到其 y 值上。此外，我们通过一次递归的 AddConst($M[x_1]$, $M[x_2]$, c) 操作，将 c 加到相应的线段上。

Min(i, j, c)。和之前一样，设 $p_p = (x_p, y_p)$ 和 $p_s = (x_s, y_s)$ 分别为数据结构中 i 的后继和 j 的前驱。设 $x_a = M[x_p]$ 和 $x_b = M[x_s]$。设 $s_{a'}$ 和 $s_{b'}$ 分别为包含 x_p 和 x_s 的线段。首先，我们更新 $s_{a'}$ 和 $s_{b'}$ 中所有（至多 4 个）需要改变其值的点。具体做法是：对 $s_{a'}$ 和 $s_{b'}$ 中满足 $x \in [i \ldots j]$ 的每个点 (x, y)，执行 Assign(min(x, Lookup(x)) x)。设 $p_1 = (x_1, y_1)$ 为 $s_{a'+1}$ 中的第一个点，$p_2 = (x_2, y_2)$ 为 $s_{b'-1}$ 中的最后一个元素，接下来只需对所有满足 $x \in [x_1 \ldots x_2]$ 的点 (x, y) 执行 Min 操作。由于表示

y 值的点 $p = (x, y)$ 为 $y = \min(D[x], R[M[x]])$，因此只需应用 $R.\text{Min}(M[x_1], M[x_2], c)$。这是由等式 $\min(\min(x, y), z) = \min(x, \min(y, z))$ 得出的。

复杂度。一次查找需要在区间加和前驱数据结构上执行 $O(1)$ 次操作，并递归调用 R。因此，$T_{\text{Lookup}}(n) = T_{\text{Lookup}}(\lceil \frac{2n}{3} \rceil) + O(\log n)$，所以查找操作的时间复杂度为 $O(\log^2 n)$。

赋值（分别为移位）操作通过在区间加和前驱数据结构上执行 $O(1)$ 次操作、执行常数次查找操作，并递归调用 R 中的赋值（分别为移位）来完成。因此，$T_{\text{Assign}}(n) = T_{\text{Assign}}(\lceil \frac{2n}{3} \rceil) + O(\log^2 n)$，所以赋值（分别为移位）的时间复杂度为 $O(\log^3 n)$。

插入、删除、AddConst 或 Min 操作通过在区间加和前驱数据结构上执行 $O(1)$ 次操作、执行常数次查找、赋值和移位操作，并递归调用 R 来完成。因此，对于所有这些操作，$T(n) = T(\lceil \frac{2n}{3} \rceil) + O(\log^3 n)$，所以它们的时间复杂度为 $O(\log^4 n)$。[9]

参考文献

1 线段树 beats。 https://codeforces.com/blog/entry/57319。2 John Aach 和 George M. Church。使用时间弯曲算法对基因表达时间序列进行比对。Bioinformatics，17(6):495–508，2001。3 Amir Abboud、Arturs Backurs 和 Virginia Vassilevska Williams。最长公共子序列及其他序列相似性度量的紧确困难性结果。载于第 56 届 FOCS，第 59–78 页，2015。4 Pankaj K. Agarwal、Kyle Fox、Jiangwei Pan 和 Rex Ying。对点序列对的动态时间规整和编辑距离进行近似计算。载于第 32 届 SoCG，第 14–18 页，2016。5 Saeed Reza Aghabozorgi、Ali Seyed Shirkhorshidi 和 Ying Wah Teh。时间序列聚类——十年回顾。Inf. Syst.，53:16–38，2015。6 Alberto Apostolico、Gad M. Landau 和 Steven Skiena。游程编码字符串的匹配。《复杂性杂志》，第 15 卷(1):4–16，1999。7 Ora Arbell、Gad M. Landau 和 Joseph S. B. Mitchell。游程编码字符串的编辑距离。Information Processing Letters，83(6):307–314，2002。8 Anthony J. Bagnall、Jason Lines、Aaron Bostrom、James Large 和 Eamonn J. Keogh。时间序列分类大比拼：近期算法进展的综述与实验评估。Data Min. Knowl. Discov.，31(3):606–660，2017。9 Karl Bringmann 和 Marvin Künnemann。字符串问题与动态时间规整的二次条件下界。载于第 56 届 FOCS，第 79–97 页，2015。10 Horst Bunke 和 János Csirik。游程编码字符串的编辑距离。载于 1992 年 ACM/SIGAPP应用计算研讨会：20 世纪 90 年代的技术挑战，第 137–143 页，1992。11 Kuan-Yu Chen 和 Kun-Mao Chao。用于计算游程编码字符串编辑距离的完全压缩算法。Algorithmica，65(2):354–370，2013。12 Raphaël Clifford、Pawel Gawrychowski、Tomasz Kociumaka、Daniel P. Martin 和 Przemyslaw Uznanski。近最优时间内计算游程编码编辑距离。载于第 44 届 MFCS，第 66:1-66:13 页，2019。13 Vincent Froese、Brijnesh J. Jain、Maciej Rymar 和 Mathias Weller。对游程编码时间序列进行快速精确动态时间规整。Algorithmica，85(2): 492-508，2023。14 Pawel Gawrychowski 和 Yanir Edri。私人通信。2016。15 Omer Gold 和 Micha Sharir。动态时间规整与几何编辑距离：突破二次时间屏障。载于第 44 届 ICALP，第 25:1–25:14 页，2017。16 Guan-Shieng Huang、Jia Jie Liu 和 Yue-Li Wang。游程编码字符串的序列比对算法。载于第 14 届 COCOON，第 319–330 页，2008。

[9] 我们没有尝试优化多对数的次数。其中一些对数因子很容易避免。

17 Youngha Hwang 和 Saul B. Gelfand。快速稀疏动态时间规整。载于第 26 届 ICPR，第 3872–3877 页，2022。18 Philip N. Klein 和 Shay Mozes。平面图优化算法。 http://planarity.org。书稿。19 William Kuszmaul。强次二次时间内的动态时间规整：低距离区域的算法与近似计算。载于第 46 届 ICALP，第 80:1–80:15 页，2019。20 William Kuszmaul。线性时间二进制动态时间规整。arXiv 预印本，2021。21 T. Warren Liao。时间序列数据聚类综述。Pattern Recognit, 38(11):1857–1874，2005。22 Jia Jie Liu、Guan-Shieng Huang、Yue-Li Wang 和 Richard C. T. Lee。游程编码字符串与未压缩字符串之间的编辑距离。Information Processing Letters, 105(1):12–16，2007。23 Alexander De Luca、Alina Hang、Frederik Brudy、Christian Lindner 和 Heinrich Hussmann。摸我一次，我就知道是你！基于触摸屏模式的隐式身份验证。载于 SIGCHI计算系统人为因素大会，第 987–996 页。ACM，2012。24 Veli Mäkinen、Esko Ukkonen 和 Gonzalo Navarro。游程压缩字符串的近似匹配。Algorithmica，35(4):347–369，2003。25 J. Mitchell。一种几何最短路径问题及其在计算游程编码字符串的最长公共子序列中的应用。技术报告，应用数学系，纽约州立大学石溪分校，纽约州，1997。26 Lindasalwa Muda、Mumtaj Begam 和 Irraivan Elamvazuthi。使用梅尔频率倒谱系数（MFCC）和动态时间规整（DTW）技术的语音识别算法。arXiv 预印本，2010。27 Abdullah Mueen、Nikan Chavoshi、Noor Abu-El-Rub、Hossein Hamooni 和 Amanda J. Minnich。Awarp：稀疏时间序列的快速规整距离。载于第 16 届 ICDM，第 350–359 页，2016。28 Abdullah Mueen、Nikan Chavoshi、Noor Abu-El-Rub、Hossein Hamooni、Amanda J. Minnich 和 Jonathan MacCarthy。加速稀疏时间序列数据的动态时间弯曲距离计算。Knowl. Inf. Syst.，54(1):237–263，2018。29 Mario E. Munich 和 Pietro Perona。用于平移不变曲线对齐的连续动态时间规整及其在签名验证中的应用。载于第 7 届 ICCV，第 108–115 页，1999。30 Saul B. Needleman 和 Christian D. Wunsch。一种适用于搜索两种蛋白质氨基酸序列相似性的一般方法。Journal of molecular biology，48(3):443–453，1970。31 Yoshifumi Sakai 和 Shunsuke Inenaga。将动态时间规整距离归约为最长递增子序列长度的更快方法。Algorithmica，84(9): 2581–2596，2022。32 Yoshifumi Sakai 和 Shunsuke Inenaga。将动态时间规整距离归约为最长递增子序列长度的一种方法。载于第 31 届 ISAAC，6:1-6:16，2020。33 Nathan Schaar、Vincent Froese 和 Rolf Niedermeier。动态时间规整下更快的二进制均值计算。载于第 31 届 CPM，28:1–28:13，2020。34 Taras K. Vintsyuk。基于动态规划的语音辨别。Cybernetics，4(1):52–57，1968。35 Xiaoyue Wang、Abdullah Mueen、Hui Ding、Goce Trajcevski、Peter Scheuermann 和 Eamonn J. Keogh。时间序列数据表示方法与距离度量的实验比较。Data Min. Knowl. Discov.，26(2):275–309，2013。36 Zoe Xi 和 William Kuszmaul。对游程编码字符串之间的动态时间弯曲距离进行近似计算。载于第 30 届 ESA，第 90:1–90:19 页，2022。37 Rex Ying、Jiangwei Pan、Kyle Fox 和 Pankaj K. Agarwal。一种简单高效的动态时间规整近似算法。载于第 24 届 ACM SIGSPATIAL，第 21:1–21:10 页，2016。38 Yunyue Zhu 和 Dennis Shasha。使用包络变换构建规整索引，以支持哼唱查询。载于第 22 届 ACM SIGMOD，第 181–192 页，2003。

缺失的证明

引理 5 的证明。我们证明 (x,y)、$(x,y+1) \in B$ 的情形，另外两种情形类似。所有进入 $(x+1,y+1)$ 的边权重均为 c。我们声称，在一个块中，数值 dist(x,y) 沿每一行和每一列都是弱单调的。这意味着，dist$(x,y) + c$（经由 (x,y) 到达 $(x+1,y+1)$ 的路径）不大于 dist$(x,y+1) + c$（经由 $(x,y+1)$ 到达 $(x+1,y+1)$ 的路径）。

为了说明数值 dist(x,y) 在一个块中沿每一行和每一列都是弱单调的，考虑同一块 B 中的两个顶点 (x,y)、$(x,y+1)$。我们证明 dist$(x,y) \le$ dist$(x,y+1)$（对称的证明可以说明 dist$(x,y) \le$ dist$(x+1,y)$）。设 P 是从 (0, 0) 到 $(x,y+1)$ 的一条最短路径。设 (x',y) 是 P 中第二个坐标为 y 的最后一个顶点。如果 $x' = x$，那么显然 dist$(x,y) \le$ dist$(x,y+1)$。否则，$x' < x$。于是，我们可以假设，P 中从 (x',y) 开始的后缀由一条对角边以及其后零条或多条垂直边组成（因为一条水平边后接一条垂直边，总不如直接使用对角边短）。现在考虑路径 P'，它从 (0, 0) 到 (x,y)，在到达 (x',y) 之前与 P 完全相同，并从 (x',y) 开始沿垂直方向继续前进。路径 P' 和 P 只有从 (x',y) 开始的后缀不同。但在这个后缀中，它们使用相同数量的 x-x' 边，且边权重相同（因为一个块中的所有边权重都相同）。这意味着 P' 和 P 长度相同，因此 dist$(x,y) \le$ dist$(x,y+1)$。

▶ **引理 23.** 对于每个 $x \in [i \dots j]$ 第 *3.1* 节中描述的过程 LeftLinearWave$(i,\ j,\ \alpha)$ 赋值 $A[x] \leftarrow \text{Min}_{t \le x}(A[t] + (x-t)\alpha)$。

证明。 设 $L(k)$ 是 $A[k]$ 经由 LeftLinearWave$(i,\ j,\ \alpha)$所赋的值，我们首先断言：

$$L(k) = \begin{cases} A[i] & k = i \\ \min(A[k], L(k-1) + \alpha) & k \in [i+1 \dots j] \end{cases}$$

我们用关于 $k - i$ 的归纳法证明这一点。对于 $k - i = 0$，LeftLinearWave$(i,\ j,\ \alpha)$ 将 $A[i] \leftarrow$赋值为$_{t \in [i \dots i]}(A[t] + (i-t)\alpha) = A[i] = L(i)$，符合要求。对于 $k - i > 0$:

$$\begin{aligned} A[k] &\leftarrow \min_{t \in [i \dots k]} (A[t] + (k-t)\alpha) = \min\Big(\min_{t \in [i \dots k-1]} (A[t] + (k-t)\alpha), A[k]\Big) \\ &= \min\Big(\min_{t \in [i \dots k-1]} (A[t] + (k-1-t)\alpha) + \alpha, A[k]\Big) = \min(L(k-1) + \alpha, A[k]) = L(k). \end{aligned}$$

根据上述内容，我们需要证明，应用 LeftLinearWave 后，$A[x] = L[x]$ 对每个$x \in [i \dots j]$ 都成立（显然，LeftLinearWave 操作不会改变 $A[x]$ 对 $x \notin [i \dots j]$的取值）。我们用关于 $x \in [i \dots j]$ 的归纳法证明这一断言。对于 $x = i$，该断言成立，因为 $L[i] = A[i]$，并且 LeftLinearWave 确实不会改变 $A[i]$。否则，假设该断言对$x - 1 \in [i \dots j-1]$成立。设 $z \in [q \dots b]$ 为最大值，使得从 $p_z =$(x_z, y_z) 开始的射线发射过程与 $x_z < x$。设 $w \in [z+1 \dots |\mathcal{P}|$ 为最小索引，使得 p_w 位于从 p_z 发射的斜率为 α 的射线 r_z 下方。设 $p' =$ $(x',\ y')$ 为ℓ_{w-1} 与从 p_z 发射的斜率为 α 的射线的交点。我们区分两种情况。

- **情况 1：** $x' \ge x-1$。在这种情况下，应用 LeftLinearWave$(i,\ j,\ \alpha)$ 后包含 $x-1$ 的线性线段是射线 r_z 的一个子线段。因此，该过程赋给 $A[x-1]$ 的值为 $r_z(x-1)$。根据归纳假设，我们有 $r_z(x-1) = L[x-1]$。对于应用更新之前的值 $A[x]$，我们分两种情况讨论。
  - $A[x] \ge L[x-1] + \alpha$。在这种情况下，点 $(x,\ A[x])$ 不在射线 r_z 下方，且 p' 必须位于 $(x,\ A[x])$ 的左侧。因此，应用该过程后，x 也位于一条线性线段上，该线段是 r_z 的子线段，因此 $A[x] = r_z(x)$。事实上，

在这种情况下，$L[x] = \text{Min}(L[x-1]+\alpha, A[x]) = L[x-1]+\alpha = r_z(x-1)+\alpha = r_z(x)$，符合要求。$A[x] < L[x-1]+\alpha$ 注意，在这种情况下，我们有 $p_w = (x, A[x])$。回想一下，当射线发射过程中遇到 p_w 时，会创建一条连接 $(x-1, r_z(x-1))$ 和 $(x, A[x])$ 的线段。因此，射线发射过程不会改变 $A[x]$ 的值。事实上，在这种情况下，我们有 $L[x] = \min(L[x-1]+\alpha, A[x]) = A[x]$。

- 情况 2：$x' < x-1$。在这种情况下，过程 LeftLinearWave(i,j,α) 没有改变 $x-1$ 的值，并且根据归纳假设，我们有 $L[x-1] = A[x-1]$。具体而言，从 q_z 发出的射线发射过程已终止，而下一个射线发射过程（如果确有必要）将从满足 $x_{z'} > x-1$ 的点 $p_{z'}$ 发出。这意味着包含 $x-1$ 的线性线段的斜率至多为 α。因此，我们必有 $A[x] < A[x-1]+\alpha = L[x-1]+\alpha$，从而得到 $L[x] = A[x]$。无论射线发射是否从 $(x, A[x])$ 开始，LeftLinearWave(i,j,α) 都不会改变 $A[x]$ 的值。如果射线发射不是从 $(x, A[x])$ 开始，那么包含 x 的线性线段不会受到 LeftLinearWave$(i.j.\alpha)$ 的影响，因为没有射线发射过程与之交互。如果射线发射过程从 $(x, A[x])$ 开始，那么应用更新后，包含 x 的线性线段将是从 $(x, A[x])$ 发出的射线 ℓ^* 的子线段，并且显然 $\ell^*(x) = A[x]$，正如所需。◀

引理的证明 14。 反设射线发射过程从被动点 $p_z \neq p_b$开始。如果 p_z 是从右侧开始射线发射的第一个点，那么 z就是 $[a \ldots b]$ 中满足 $\alpha_{z-1} < -\alpha$的最大索引。但由于 p_z是被动点，我们有$-\alpha > \alpha_{z-1} \geq \alpha_z$，这与 z的最大性矛盾（注意 $p_z \neq p_b$，因此 $z+1 \in [a \ldots b]$）。

否则，令 p_q 为在 p_z 之前最后一个发生过射线发射过程的点。令 $p_{q'}$ 为从 p_q发射的射线下方的第一个点。由于 p_z 是下一个发射射线的点，z是 $[a \ldots q']$ 中满足 $\alpha_{z-1} < -\alpha$的最右侧点。由于 p_z是被动点，我们有$-\alpha > \alpha_{z-1} \geq \alpha_z$。如果 $z \neq q'$，则有 $z+1 \in [a \ldots q']$，这与 z的最大性矛盾。否则，$p_z = p_{q'}$ 是从 p_q发射的斜率为 $-\alpha$ 的射线下方的第一个点。由 $\alpha_z < -\alpha$ 可知，p_{z+1} 也在射线下方，这与 $p_z = p_{q'}$ 是射线下方的第一个点矛盾。