连续出现的压缩索引*

Paweł Gawrychowski[†1]、Garance Gourdel[‡2]、Tatiana Starikovskaya[‡3]和 Teresa Anna Steiner[§4]

[1]计算机科学研究所，弗罗茨瓦夫大学，gawry1@gmail.com[2]DI/ENS，PSL研究大学，IRISA Inria Rennes，garance.gourdel@irisa.fr[3]DI/ENS，PSL研究大学，tat.starikovskaya@gmail.com[4]DTU Compute，terst@dtu.dk

arXiv:2304.00887v1 [cs.DS] 3 Apr 2023

摘要

字符串算法中的一个基本问题是索引，即针对特定查询对给定字符串进行预处理。迄今为止，对于查询给定模式 P 的精确出现位置问题，我们已有多种高效的解决方案。然而，实际应用促使我们考虑更复杂的查询，例如查询两个模式的近似出现位置。最近，Bille 等人 [CPM 2021]提出了这类查询的一种变体，称为带间隔的连续出现：查询由两个模式 P_1 和 P_2 以及一个范围 $[a, b]$组成，需要找出 P_1 和 P_2 的所有连续出现位置（q_1，q_2），满足条件 $q_2 - q_1 \in [a, b]$。根据他们的结果，即使 $a = 0$ 固定，我们也无法期望为这类查询构建非常高效的索引结构（尽管他们同时给出了一个非平凡的上界）。受此启发，我们聚焦于以直线程序（SLP）表示的文本，并设计了一种索引，其空间占用相对于文法大小为多项式级，且能以除多对数因子外最优的时间回答此类查询。

*这是被 CPM 2023 接收的论文的完整版本。

†本研究部分由波兰国家学术交流局的 Bekker计划（PPN/BEK/2020/1/00444）以及法国国家研究署（ANR）的 ANR-20-CE48-0001 项目资助。

‡部分经费由法国国家研究署（ANR）提供的资助项目 ANR-20-CE48-0001 支持。§经费由 VILLUM FONDEN 提供的研究资助（VIL51463）支持。

1 引言

索引问题的目标是对字符串进行预处理，以定位给定模式的出现位置。对于长度为 N 的字符串，后缀树 [36] 或后缀数组 [31],等结构占用线性于 N 的空间，并能在与模式长度 m 成线性关系的时间内回答此类查询。迄今为止，我们已有多种在空间和时间效率方面表现优异的解决方案（无论在理论上还是实践中）。我们请读者参阅 Lewenstein [29] 撰写的优秀综述，其中概述了若干相关方法及其扩展，并着重介绍了它们与正交范围查询之间的联系。

然而，从潜在应用的角度来看，理想情况下应允许比仅在预处理后的文本中定位给定模式的精确匹配更一般的查询，同时保持查询时间关于预处理字符串的长度为次线性。一个非常一般的查询是定位与正则表达式匹配的子串。最近，Gibney 和 Thankachan [19]证明，如果在线矩阵-向量乘法猜想成立，那么即使预处理时间为多项式级，也无法在次线性时间内回答正则表达式查询。一个更合理但仍颇有意义的查询，可以关注两个给定模式之间距离最近的出现位置，或仅要求它们彼此足够接近。

针对两个模式相关查询的字符串预处理，最早是在文档检索的背景下进行研究的，其目标是预处理一个字符串集合。在双模式文档检索问题 中，查询包含两个模式 P_1 和 P_2，我们必须报告同时包含二者的所有文档 [32]。在禁用模式查询问题 中，我们必须报告所有包含 P_1 但不包含 P_2 [15]的文档。对于这两个问题，渐近意义上最快的线性空间解答在回答查询时都需要 $\Omega(\sqrt{N})$ 时间，其中 N 是所有字符串的总长度 [23, 22]。也就是说,¯复杂度高度依赖于字符串的长度。Larsen 等人 [28] 揭示了布尔矩阵乘法与这两个问题之间的联系，从而有条件地解释了较高的 $\Omega(\sqrt{N})$ 查询复杂度。随后，Kopelowitz 等人 [27] 利用与 3SUM问题的联系，给出了更有力的论证。与本文更相关的是 Kopelowitz 和 Krauthgamer [26], 探讨的问题：预处理一个字符串，以便给定两个模式 P_1 和 P_2后，计算它们彼此距离最近的出现位置。他们论文的主要结果是一种数据结构，可在 $O(N^{1.5}\log^{\epsilon} N)$时间内构建，并能在$O(|P_1|+|P_2|+\sqrt{N}\log^{\epsilon} N)$时间内回答此类查询；该字符串的长度为 N，且对任意 $\epsilon > 0$均成立。他们还揭示了布尔矩阵乘法与此问题之间的联系，指出要同时从预处理时间和查询时间中消除 $O(\sqrt{N})$项存在困难。

—

本文关注的是近期提出的一种索引问题变体，称为连续出现的间隔索引。在该问题中，查询由两个模式 P_1 和P_2 以及一个范围 $[a,b]$组成，目标是找出 P_1, P_2 的连续出现位置对，且两者之间的距离落在范围 $[a,b]$内。Navarro 和 Thankanchan [33] 证明，对于 $P_1 = P_2$，存在一个占用 $O(n\log\ n)$ 空间的索引，其查询时间最优，为 $O(m+\text{occ})$，其中 $m = |P_1| = |P_2|$, occ 是需要报告的位置对数。不过，他们在结论中指出，将其方案扩展到由两个模式构成的一般情形，可能并不可行。Bille 等人 [4] 提供了该一般情形具有困难性的证据，并通过将其复杂度与集合交集问题的复杂度联系起来，确立了连续出现的间隔索引的一个（条件性）下界。˜这一界表明，至少对于大小为 $O(N)$ 的索引，若查询时间优于 $\tilde{O}(|P_1|+|P_2|+\sqrt{N})$，就会与集合不相交猜想相矛盾,¯即使 $a = 0$ 固定也是如此。尤其是，查询时间主要取决于模式长度（或许再加上一些对数因子）的方案——这可以说是字符串索引的核心目标——在此情形下不太可能实现。

受连续出现的间隔索引（条件）下界的启发，

我们考虑这一问题针对查询区间 $[0, b]$ 的压缩版本。对于精确模式匹配，长期以来，研究者一直致力于设计所谓的压缩索引，即其大小取决于文本压缩表示长度的索引结构；例如，参见《算法百科全书》 [30] 或《数据库系统百科全书》 [13]中的相关条目。这引出了以下研究方向：我们能否设计一种高效的、用于连续出现的压缩间隙索引?

答案当然取决于所选的压缩方法。为了设计一个占用空间极少的索引，我们聚焦于最具挑战性的情形：压缩方法能够描述长度呈指数级增长（相对于其表示规模）的字符串。直线程序（SLP）是一种优雅的压缩方法形式，它是一种恰好描述一个字符串的上下文无关文法。众所周知，SLP 能够以对数因子级的额外开销捕获流行的 Lempel–Ziv 压缩方法 [7, 35], ，同时还提供了更便捷的接口；尤其是，它支持在 O（log N）时间内进行随机访问 [5]。

目前已知，模式匹配可以在 SLP 压缩空间中通过索引高效完成。假设一个长度为 N 的字符串 S 由包含 g 个产生式的 SLP 描述，Claude 和 Navarro [9] 为 S 设计了一个占用 $O(g)$ 空间的索引，使得能够在 $O(m^2 \log\log N + \text{occ} \log g)$ 时间内检索长度为 m 的模式的所有出现位置。最近，有若干结果在使用可比的 $O(g \log N)$ 空间的同时，改进了查询时间上界：Claude、Navarro 和 Pacheco [10] 展示了一个查询时间为 $O((m^2 + \text{occ}) \log g)$ 的索引；Christiansen 等人 [8] 利用字符串吸引子将时间上界进一步改进至 $O(m + \text{occ} \log^{\epsilon} N)$；Díaz-Domínguez 等人 [12] 则实现了 $O((m\log m + \text{occ}) \log g)$ 的查询时间。

然而，高度可压缩的字符串并不总是更容易预处理。负面来看，Abboud 等人 [1] 证明，对于压缩字符串上的某些问题，例如计算最长公共子序列，无法完全避免对未压缩字符串长度的高度依赖；而对于压缩字符~~串上的其他问题，例如上下文无关文法解析或 RNA 折叠，基本上无法指望比解压字符串~~并处理其未压缩表示更好的方法！线性代数中的某些相关问题 [2]也存在同样的情况。因此，我们尚不清楚，在连续出现的间隔索引问题中，是否能够避免对未压缩字符串长度的高度依赖。

在这项工作中，我们回应了 Bille 等人 [4] 提出的下界，并证明，尽管 Abboud 等人 [1], 给出了负面结果，但假设文本具有很高的可压缩性，我们仍能绕过这一限制：

定理 1.1。对于大小为 g、表示长度为 N的字符串 S的直线程序，存在一个 $O(g^5 \log^5 N)$ 空间的数据结构，可支持以下查询：给定两个长度均为 $O(m)$ 的模式 P_1和 P_2，以及一个范围 $[0, b]$，报告 P_1 和 P_2 之间所有相距 $d \in [0, b]$的 occ 个连续出现位置。查询时间为 $O(m \log N + (1 + \text{occ}) \cdot \log^4 N \log\log N)$。

虽然达到 $O(g)$ 的空间复杂度和 $O(m + \text{occ})$ 的查询时间，会根据 Bille 等人 [4], 的归约与集合不相交猜想相矛盾，但我们或许会想知道，能否在不增加查询时间的情况下改进空间复杂度，以及当 a 不固定时，该问题的真正复杂度是什么（回想一下，$[a, b]$ 是用于限制待报告共现之间距离的范围）。虽然我们将空间复杂度改进和一般情形留作有趣的开放问题，但我们证明，在较简单的情形 $a = 0$，$b = N$（即 P_1 和 P_2 的起始位置之间的距离不受限制时），我们的方法确实能达到 $O(g^2 \log^4 N)$ 的空间复杂度，见推论 3.2[1]。

全文假设采用字长为$\Theta(\log N)$的单位成本 RAM 计算模型。所有空间复杂度均指数据结构所使用的字数。

[1]请注意，Bille 等人 [4] 给出的条件下界并不适用于这一较简单的情形。

2 预备知识

长度为 $|S| = N$的字符串 S是由字母表 Σ中的字符组成的序列 $S[0]S[1]\dots S[N-1]$。我们用 rev(S) 表示 S的反转 $S[N-1]S[N-2]\dots S[0]$。若 $i \le j$，我们定义 $S[i\dots j]$ 等于$S[i]\dots S[j]$，并称其为 S的子串；否则定义其为空串。我们还使用记号 $S[i\dots j)$和 $S(i\dots j]$，它们自然分别表示 $S[i]\dots S[j-1]$和 $S[i+1]\dots S[j]$。我们称子串 $S[0\dots i]$是 S的前缀，并使用简化记号 $S[\dots i]$；称子串 $S[i\dots N-1]$是 S的后缀，记作 $S[i\dots]$。若对于某个 $0 \le i \le j \le N-1$，有 $X = S[i\dots j]$，则称 X是S的子串。索引 i称为 S中 X的出现位置。

q_1 的一个出现位置与 P_1 的一个出现位置构成字符串 P_1的连续出现（共现）P_2，当且仅当在字符串 S 中，q_1与 q_2之间不存在 P_1的出现位置，即形式化地说，在（q_1，q_2] 中不存在 P_1 的出现位置，并且在$[q_1, q_2$中不存在 P_2 的出现位置）。为简洁起见，如果 $q_2 - q_1 \le b$，我们称共现是 b-接近的。

如果对于所有 $i = 0$，.....，$N-1-\pi$，$S[i] = S[i+\pi]$，则整数 π是长度为 N的字符串 S的周期。字符串 S 的最小周期称为 S的周期。如果 S 的周期至多为 $N/2$，则称 S 是周期性的。我们利用 Fine–Wilf 周期性引理的一个众所周知的推论 [14]:

推论 2.1. 如果字符串 Y 在字符串 X 中至少出现三次，其中 $|X| \le 2|Y|$，那么 Y 在 X 中的出现位置构成一个等差数列，其公差等于 Y 的周期。

2.1 文法

定义 2.2 (直线程序 [25])。直线程序 ***(SLP)*** G 是一种上下文无关文法 ***(CFG)***，由一组非终结符、一组终结符、一个初始符号和一组产生式组成，并满足以下性质：

- 一个产生式由左部和右部组成，其中左部是一个非终结符 A，右部则是一个序列 BC，其中 B、C 是非终结符；或者是一个终结符；
- 每个非终结符恰好是一个产生式的左部；
- 存在一种线性序 $<$，定义在非终结符上，使得 $A < B$ 每当 B 出现在与 A关联的产生式的右部中时。

一种游程直线程序 (RLSLP) [34] 还允许形如$A \to B^k$ 的产生式，其中 k为正整数；这对应于将 B拼接 k 个副本。若 A 对应于产生式 $A \to a$，其中 a 是终结符，则记 head(A) 为 $= a$，tail(A) 为 $= \varepsilon$（空串）；若 A 对应于产生式 $A \to BC$，则记 head(A) 为 $= B$，tail(A) 为 $= C$；最后，若 A 对应于产生式 $A \to B^k$，则 head(A) 为 $= B$，tail(A) 为 $= B^{k-1}$。

序列（由终结符和非终结符组成）S 的展开 S 是通过反复将非终结符替换为相应产生式的右部，直至只剩下终结符而得到的字符串。我们称 G 表示 其初始符号的展开。

定义 2.3（解析树）. 直线程序（*RLSLP*）的解析树 定义为如下有根树：

- 根节点由初始符号标记；
- 每个内部节点都标记有一个非终结符；

- 如果 S 是初始符号的展开，那么解析树的第 i 个叶节点标记有一个终结符 $S[i]$;

- 标记为非终结符 A 且与产生式 $A \to BC$关联的节点，其中B、C 均为非终结符，有 2 个子节点，分别标记为 B 和 C。如果 A 与产生式 $A \to a$关联，其中 a 是终结符，那么该节点有一个子节点，标记为 a。

- （仅限 *RLSLP*）标记为非终结符 A 且与产生式$A \to B^k$关联的节点，其中 B 是非终结符，有 k 个子节点，每个子节点都标记为 B。

文法的大小 是其产生式的数量。文法的高度 是解析树的高度。我们称非终结符 A 是非终结符 B 的祖先，如果解析树中存在节点 u, v，分别标记为 A, B，且 u是 v的祖先。对于解析树中的节点u，用 $\mathsf{off}(u)$ 表示其根节点为 u的子树左侧的叶节点数量。

定义 2.4（相关出现位置）。设 A 是与产生式$A \to$ 头部（A）尾部（A）相关联的非终结符。若字符串 P 在 A 中的出现位置 q 满足 $q = |\text{头部}(A)| - s \le |\text{头部}(A)| \le \overline{q} + |P| - 1$，则称其为具有分割点 的相关出现位置。

例如，在图 1 中，$P = cab$ 在 C 中的出现位置 $q = 3$ 是具有分割点 $s = 1$ 的相关出现位置，但 A 中不含 P的相关出现位置。

断言 2.5。设 q 是字符串 P 在字符串 S中的一个出现位置。考虑表示 S的 *RLSLP* 的解析树，并设 w 是其子树中包含叶节点 $S[q]$、$S[q+1]$、.....、$S[q+|P|-1]$ 的最低节点，则以下情况之一成立：

1. w 的标签 A 与产生式 $A \to BC$相关联，且 $q - \mathsf{off}(w)$ 是 A中的相关出现位置；或者
2. w 的标签 A 与产生式 $A \to B^r$ 相关联，且对于某个 $0 \le r' \le r$，$q - \mathsf{off}(w) = q' + r'|B|$，其中 q' 是 A中 P 的相关出现位置。

证明。首先假设 A 与产生式 $A \to BC$相关联。于是，以 w的左子节点为根的子树（对应于 B）不包含 $S[q+|P|-1]$，以 w 的右子节点为根的子树（对应于 C）不包含 $S[q]$。因此，$q - \mathsf{off}(w)$ 是 A中的一个相关出现位置。

现在考虑 A 与产生式 $A \to B^r$相关联的情形。标记为 $S[q]$ 和 $S[q+|P|-1]$ 的叶节点属于以 A的不同子节点为根的子树。如果 $S[q]$ 属于以 A的第（$r'+1$）个子节点为根的子树，那么 $q' = q - \mathsf{off}(w) - |B| \cdot r'$ 是 A中 P 的一个相关出现位置。 □

定义 2.6 (分割点)。考虑 *RLSLP* G的一个非终结符 A。若其与产生式 $A \to BC$相关联，则定义

$$\mathrm{Splits}(A, P) = \mathrm{Splits}_{\mathrm{rev}}(A, P) = \{s : \ q\text{是 } A \text{ 中以分割点 } s\} \text{ 分割的 } \overline{P} \text{的相关出现}.$$

如果 A 与规则 $A \to B^k$ 相关联，则定义

$$\mathrm{Splits}(A, P) = \{s : \ q\text{是 } A \text{ 中以分割点 } s\} \text{ 分割的} \overline{P} \text{ 的相关出现；} \mathrm{Splits}_{\mathrm{rev}}(A, P) = \{|P| - s : \ q\text{是}\mathrm{rev}(P)\text{在}\mathrm{rev}(A)\text{中的以分割点 } s\}. \text{分割的相关出现}$$

定义 $\mathsf{Splits}(G, P)$（$\mathsf{Splits}_{\mathrm{rev}}(G, P)$）为以下集合的并集：对所有非终结符 A属于 G，以及 $\mathsf{Splits}'(G, P) = \mathsf{Splits}(G, P) \cup \mathsf{Splits}_{\mathrm{rev}}(G, P)$。

我们需要以下引理，它可以由 Gawrychowski 等人推导得出。 [18]:

引理 *2.7.* 设 G 是一个大小为 g、表示长度为 N 的字符串 S 的直线程序，其中 $g \le N$。存在一种拉斯维加斯算法，可在时间 $O(g \log N)$ 内以高概率构造一个大小为 $g' = O(g \log N)$、高度为 $h = O(\log N)$ 的 *RLSLP* G' 来表示 S。该 *RLSLP* 还具有以下性质：对于长度为 m 的模式 P，我们可以在 $O(m \log N)$ 时间内提供一份证书，证明 P 不在 S 中出现；或者计算集合 $Splits'(G', P)$。在后一种情况下，$|Splits'(G', P)| = O(\log N)$。

2.2 紧凑字典树

我们假设读者熟悉紧凑字典树的定义（例如，参见 [21]）。形式上，字典树是一棵表示按字典序排列的字符串集合的树。字典树的边以字符串作为标签。我们将节点 u 的标签 $\lambda(u)$ 定义为从根节点到 u 路径上各标签的拼接，并将区间 $I(u)$ 定义为以 $\lambda(u)$ 开头的字符串集合所对应的区间。从实现角度来看，我们假设节点 u 由区间 $I(u)$ 指定。字符串 P 的位置 是满足P 为 $\lambda(u)$ 前缀的最小深度节点 u 。

对于一般的字符串集合 $\mathcal{S} = \{S_1, \ldots, S_k\}$，基于标准树结构的字典树实现需要 $\Theta\left(\sum_{i=1}^{k} |S_i|\right)$ 空间。给定一个长度为 m 的模式 P 及 $\tau > 0$ 个后缀 Q_1、.....、Q_τ（属于 P），字典树允许在 $O(m^2)$ 时间内检索（按字典序排序的） $\mathcal{S}$ 中以Q_1、.....、Q_τ 为前缀的字符串范围。然而，在这项工作中，我们仅为非常特殊的字符串集合构建字典树，因此可以基于 Christiansen 等人的技术实现得高效得多。 [8],证明见附录 A:

引理 2.8。 给定一个大小为 g 、高度为 h 的 *RLSLP* G 。假设集合 $\mathcal{S}$ 中的每个字符串要么是 G 中某个非终结符展开后的前缀或后缀，要么是其反转。 $\mathcal{S}$ 的字典树可用 $O(|\mathcal{S}|)$ 空间实现，并以 $O(m + \tau \cdot (h + \log m))$ 时间支持以下查询：给定一个长度为 m 的模式 P，以及 P 的后缀 Q_i，对每个 i，查找按字典序排列的 $\mathcal{S}$ 中以 Q_i 为前缀的字符串区间。

3 非终结符中的相关出现位置、极值出现位置和前驱出现位置

本节介绍一种数据结构，它支持多种高效查询，这些查询将用于证明定理 1.1。我们还说明如何将其用于连续出现这一较简单情形下的索引（$a = 0$， $b = N$）。回顾一下，文本 S 是一个长度为 N 的字符串，由大小为 g 的直线程序 G 表示。应用引理 2.7 后，我们将 G 转换为大小为 $g' = O$（$g \log N$）、深度为 $h = O(\log N)$ 的 RLSLP G'，用以表示 S；从现在起，我们固定使用该 RLSLP。首先说明如何以较小的空间处理 G' ，以支持多种高效查询：

定理 3.1. 存在一种空间复杂度为 $O(g^2 \log^4 N)$ 的数据结构，对于 G'，给定一个长度为 m 的模式 P，可在 $O(m \log N + \log^2 N)$ 时间内对其进行预处理，以支持针对 G' 的给定非终结符 A 执行以下查询：

1. 在 A 中报告 P 的相关出现，并按排序顺序排列，时间为 $O(\log N)$;
2. 在 A 中判定 P 是否出现，时间为 $O(\log N \overline{\log} \log N)$;
3. 在 A中报告 P 的最左和最右出现位置，包括头部(A)和尾部(A) ，$\overline{\text{时间复杂度为}O(\overline{\log^2 N} \log \log N)}$ 。

4. 给定位置 p，在 A 中查找 P 的最右（最左）出现位置 $q \le p$ （$q \ge p$），时间复杂度为$\overline{O}$ $(\log^3 N \log\log N)$（前驱/后继）。

在继续证明之前，我们先推导一个数据结构，用于报告给定一对模式的所有连续出现位置（共同出现位置）。

推论 3.2. 对于一个大小为 g、表示长度为 N 的字符串 S 的直线程序，存在一个占用 $O(g^2 \log^4 N)$ 空间的数据结构，支持以下查询：给定两个长度均为 $O(m)$ 的模式 P_1、P_2，报告 S 中 P_1 和 P_2 的所有 occ 共同出现位置。查询时间为 $O(m\log N + (1+\text{occ}) \cdot \log^3 N \log\log N)$。

证明。我们利用定理 3.1 的数据结构来处理 G'。为了报告 S 中 P_1、P_2 的所有共同出现位置，我们用 $(m \log N + \log^2 N)$ 时间预处理 P_1、P_2 在 O 中的信息，然后按如下方式进行。假设我们要在字符串 $S[i\ldots]$ 中查找 P_1 和 P_2 的最左共同出现位置，且初始时 $i = 0$。我们通过对 G' 的初始符号（其展开为整个字符串 S）进行后继查询，找到 P_1 的最左出现位置 q_1'，并要求其满足 $q_1' \ge i$（若存在）。然后，我们通过一次后继查询找到满足 $q_2 \ge q_1'$ 的 P_2 的最左出现位置 q_2（若存在），并通过一次前驱查询找到满足 $q_1 \le q_2$ 的 P_1 的最右出现位置 q_1。如果 q_1' 或 q_2 不存在，那么 $S[i\ldots]$ 中就没有更多共同出现位置。否则，显然，(q_1, q_2) 是一个共同出现位置，并且不存在以 $S[i\ldots q_2]$ 开始的其他共同出现位置。在这种情况下，我们返回 (q_1, q_2) 并设置 $i = q_2 + 1$。检索阶段的运行时间为 $O(\log^3 N \log\log N \cdot (\text{occ}+1))$，因为我们最多使用三次后继/前驱查询，即可输出一个新的共同出现位置，或确定不存在更多共同出现位置。

□

3.1 定理 3.1 的证明

该数据结构由两个紧凑字典树 T_{pre} 和 T_{suf} 构成，定义如下。对于每个非终结符 A，我们在 T_{pre} 中存储 $\text{rev}(\text{head}(A))$，并在 T_{suf}中存储 $\overline{\text{tail}(A)}$。我们通过计算它们的重路径分解来扩充 T_{pre} 和 T_{suf}：

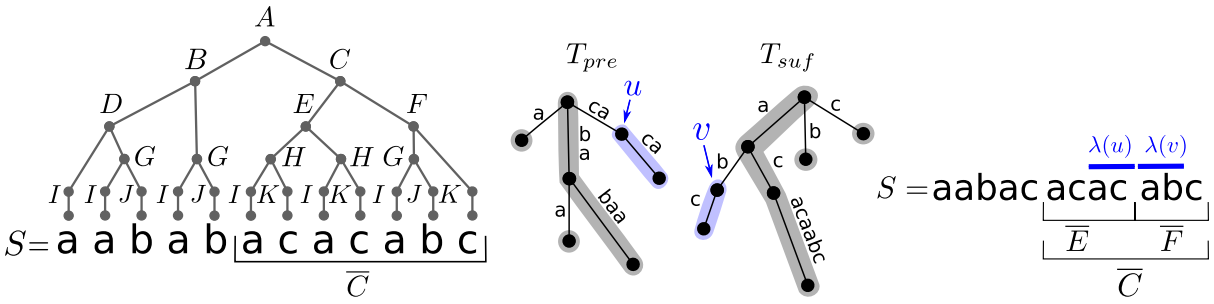
定义 3.3。 字典树 T 的重路径 是这样一条路径：从 T 的根节点出发，在路径上的每个节点 v 处，都沿着其子树中叶节点数最多的子节点（重子节点）分支；若并列，则任意选择。重路径分解是递归定义的一组互不相交的路径，即它由包含T 的重路径的单元素集合，以及 T 中从重路径分支出去的子树的重路径分解的并集构成。

对于 G'的每个非终结符 A、T_{pre}中的一条重路径 h_{pre} 以及 T_{suf}中的一条重路径 h_{suf}，我们构造一个点的多重集 $\mathcal{P}(A, h_{pre}, h_{suf})$。对于每个非终结符 A' 以及节点 $u \in h_{pre}$、$v \in h_{suf}$，当且仅当 A'，u，v 满足以下性质时，多重集包含一个点（$|\lambda(u)|$，$|\lambda(v)|$）：

1. A 是 A'的祖先；
2. $I(u)$ 包含 $\text{rev}(\overline{\text{head}(A')})$，而 $I(v)$ 包含 $\overline{\text{tail}(A')}$。
3. u，v 分别是满足性质 2 的 h_{pre}，h_{suf}中的最低节点。

（见图 1。）集合 $P(A, h_{pre}, h_{suf})$ 存储在双侧二维正交范围空性数据结构 [29, 6] 中，该数据结构占用 $O(|\mathcal{P}(A, h_{pre}, h_{suf})|)$ 的空间。给定一个形如$[\alpha, \infty] \times [\beta, \infty]$的二维范围，该数据结构可在$O(\log\log N)$ 时间内判断该范围是否包含 $\mathcal{P}(A, h_{pre}, h_{suf})$ 中的点。

断言 3.4. 该数据结构占用 $O(g^2 \log^4 N)$ 空间。



(a) G'的解析树。 (b) 使用分割点 $s = 1$ 搜索 cab。

图 1：SLP G' 生成字符串 S =aababacacabc。节点 u 和 v 分别是 T_{pre} 和 T_{suf} 中 c 和 ab 的位置。T_{pre} 中的重路径 h_{pre} 和 T_{suf} 中的重路径 h_{suf} 以蓝色标出。我们有 $(2, 2) \in \mathcal{P}(A, h_{pre}, h_{suf})$，对应于 C, u, v。

证明。每个非终结符 A' 至多有 g' 个不同的祖先，并且 T_{pre}或 T_{suf} 中的每条根到叶路径都会经过 $O(\log g')$ 条重路径（因为每次切换重路径时，当前节点子树中的叶节点数量至少减半）。因此，每个非终结符在所有正交范围空性数据结构中总共创建 $O(g' \log^2 g') = O(g \log^3 N)$ 个点。

□

当收到模式 P 时，我们通过引理 2.7 在 $O(m \log N)$ 时间内计算 Splits′(G', P)，或提供一份证书，证明 P 不出现在 S 中；此时，G' 的非终结符展开中不会出现 P。回顾一下，$|\text{Splits}'(G', P)| \in O(\log N)$。然后，我们在 $O(\log^2 N)$ 时间内对 Splits′(G', P) 排序（这是一个技术细节，可使我们无需额外时间开销即可按序报告相关出现位置）。最后，对于每个 $s \in$ Splits′(G', P)，我们计算 T_{pre} 中以 rev$(P[\ldots s])$ 为前缀的字符串区间（即 T_{pre} 中 rev$(P[\ldots s])$ 的位置对应的区间 $I(u)$），以及 T_{suf} 中以 $P(s\ldots]$ 为前缀的字符串区间（即 T_{suf} 中 $P(s\ldots]$ 的位置对应的区间$I(u)$ u）。根据引理 2.8，利用 $\tau = |\text{Splits}'(G', P)| = O(\log N)$ 和 $h = O(\log N)$，此步骤的时间复杂度为 $O(m + \log^2 N)$。

报告相关出现很简单：根据定义，每个 A 中 P 的相关出现位置 q |都等于某个 $s \in$ Splits′(G', P) 的 $|-s\,\text{head}(A)| - s$，其中 rev$(P[\ldots s])$ 是 rev(head(A)) 的前缀，且 $P(s\ldots]$ 是 tail(A) 的前缀。我们已经知道 T_{suf} 和 T_{pre} 中以 rev$(P[\ldots s])$ 和 $P(s\ldots]$ 开头的字符串的区间，因此这两个条件都可以针对每个分割点在常数时间内检查；总体耗时为 $O(|\text{Splits}'(G', P)|) = O(\log N)$。注意，由于 Splits′$(G', P)$ 已排序，报告的相关出现位置也会按顺序排列。

现在我们来说明如何对一个非终结符回答空性查询：

断言 3.5。设 A 是一个非终结符，它标记解析树中 G' 的一个节点。我们可以在 $O(\log N \log\log N)$ 时间内判定 A 是否包含 P 的一个出现位置。

证明。下面我们证明，P在 A中出现，当且仅当存在一个分割点 $s \in$Splits′(G', P)，使得对于 u（即 rev$(P[\ldots s])$ 在 T_{pre}中的位置）以及 v（即 $P(s\ldots]$在 T_{suf}中的位置），对于 h_{pre}（即 u 在 T_{pre}中所在的重路径）和 h_{suf}（即 v 在 T_{suf}中所在的重路径），矩形$[|\lambda(u)|, +\infty] \times [|\lambda(v)|, +\infty]$包含来自 $\mathcal{P}(A, h_{pre}, h_{suf})$的一个点。在继续证明之前，请注意，根据 $|\text{Splits}'(G', P)|$ 的界，我们可以通过 O次范围空性查询来判断 P是否在 A中出现，因此查询时间为 $O(\log N \log\log N)$。

假设 $[|\lambda(u)|, +\infty] \times [|\lambda(v)|, +\infty]$ 包含一个点 $(x, y) \in \mathcal{P}(A, h_{pre}, h_{suf})$，该点对应于非终结符 A'。根据构造，A 是 A' 的祖先，u 的子树包含一个对应于 rev(head(A')) 的叶节点，而 v 的子树包含一个对应于 tail(A') 的叶节点。因此，A' 包含 P 的一个出现位置，这意味着 A 包含一个出现位置

P。为证明反向结论，令 $\ell = \mathsf{off}(u) + 1$ 且 $r = \mathsf{off}(u) + |A|$，即 $S[\ell \dots r] = A$。字符串 A 包含 P 的一个出现位置 $A[q \dots q + |P|)$，当且仅当 $S[\ell + q \dots \ell + q + |P|)$ 是 S 中 P 的一个出现位置。由断言 2.5 可知，若 w 是 G' 的解析树中最低的节点，且其子树包含叶节点 $S[\ell+q], \dots, S[\ell+q+|P|-1]$，其标签为 A'，那么存在一个分割点 $s \in \mathsf{Splits}'(G', P)$，使得 $\mathsf{rev}(P[\dots s])$ 是 $\mathsf{rev}(\mathsf{head}(A'))$ 的前缀，且 $P(s \dots]$ 是 $\mathsf{tail}(A')$ 的前缀。根据 u 和 v 的定义，标记为 $\mathsf{rev}(\mathsf{head}(A'))$ 的 T_{pre} 叶节点属于 $I(u)$，而标记为 $\mathsf{tail}(A')$ 的 T_{suf} 叶节点属于 $I(v)$。令 $h_{pre}(h_{suf})$ 为 $T_{pre}(T_{suf})$ 中包含 $u\,(v)$ 的重路径，(x, y) 为在 $\mathcal{P}(A, h_{pre}, h_{suf})$ 中为 A' 创建的点。由于 $|\lambda(u)| \le x$ 且 $|\lambda(v)| \le y$，矩形 $[|\lambda(u)|, +\infty] \times [|\lambda(v)|, +\infty]$ 非空。 □

接下来说明如何检索非终结符中的最左/最右出现位置，以及如何回答前驱/后继查询。这四类查询的核心思路都是从解析树中任意一个标记为 A 的 G' 节点开始，通过空性查询和情况分析递归向下。由于每次从非终结符递归到其子节点时，展开的长度都会减小，且 G' 的高度为 $h = O(\log N)$，因此可以达到所需的查询时间。完整细节见附录 B。

4 近距离共现的压缩索引

本节给出我们的主要结果，即定理 1.1。回顾一下，S 是一个长度为 N 的字符串，由大小为 g 的直线程序 G 表示。我们首先应用引理 2.7，将 G 转换为大小为 $g' = O(g \log N)$、高度为 $h = O(\log N)$ 的 RLSLP G'，以表示 S。

查询算法采用以下策略：首先，识别 G' 的所有非终结符，使得它们的展开包含一个 b 近距离相关共现，其中相关共现的定义与相关出现位置类似：

定义 4.1 (相关共现)。设 A 是 G' 的一个非终结符。若 P_1, P_2 在 A 中的一个共现 (q_1, q_2) 满足 $q_1 \le |\text{头部}(A)| \le q_2 + |P_2| - 1$，则称其为相关共现。

其次，在这些非终结符中分别检索所有 b 近距离相关共现；最后，通过遍历 G' 的（剪枝后）解析树，报告所有 b 近距离共现。根据以下论断，这是可行的：

论断 4.2。假设 P_2 不是 P_1 的子串，并设 (q_1, q_2) 是字符串 S 中的一个共现。在 G' 的解析树中，存在唯一的节点 u，使得以下条件之一成立

1. 其标签 A 与产生式 $A \to BC$ 相关联，且（$q_1 - \mathsf{off}(u)$ $q_2 - \mathsf{off}(u)$）是 P_1、P_2 在 A 中的一个相关共现；2. 其标签 A 与产生式 $A \to B^k$、$q_1 - \mathsf{off}(u) = q_1' + k'|B|$、$q_2 - \mathsf{off}(u) = q_2' + k'|B|$ 相关联，其中 $0 \le k' \le k$ 为某个值，且（q_1'、q_2'）是 P_1、P_2 在 A 中的一个相关共现。

证明。设 A 为解析树中包含叶节点 $S[q_1]$、$S[q_1+1]$、……、$S[q_2+|P_2|-1]$ 且其子树中含有这些叶节点的最低节点 u 的标签。由于 P_2 不是 P_1 的子串，因此 A 不能与产生式 $A \to a$ 相关联。根据定义，$S[\mathsf{off}(u) + 1]$ 是该节点子树中的最左叶节点。

首先，假设 A 与产生式 $A \to BC$ 相关联。于是，以 u 的左子节点（标记为 B）为根的子树不包含 $S[q_2 + |P_2| - 1]$，且以 u 的右子节点（标记为 C）为根的子树不包含 $S[q_1]$。因此，$(q_1 - \mathsf{off}(u)\ q_2 - \mathsf{off}(u))$ 是 A 中 P_1，P_2 的一个相关共现。

现在考虑 A 与产生式 $A \to B^k$ 相关联的情况。标记为 $S[q_1]$ 和 $S[q_2 + |P_2| - 1]$ 的叶节点属于以 A 的不同子节点为根的子树。如果 $S[q_1]$

属于以 A 的第 k' 个子节点为根的子树，那么 $(q_1 - \mathrm{off}(u) - |B| \cdot (k'-1), q_2 - \mathrm{off}(u) - |B| \cdot (k'-1))$ 是 P_1, P_2 在 A 中的相关共现。 □

4.1 组合观察

非正式地，我们定义一个由 $O(g^2)$ 个字符串组成的集合，并证明对于任意模式 P_1, P_2，集合中都存在两个字符串 S_1, S_2，满足以下性质：每当 A 中某个非终结符的展开包含一对出现位置 P_1, P_2，且它们构成相关共现时，邻近位置上就存在 S_1, S_2 的出现位置。这样，我们就可以预处理 G' 中非终结符内集合中各字符串的出现位置，并利用它们检测 P_1, P_2 的距离不超过 b 的相关共现。

考虑两棵字典树 T_{pre} 和 T_{suf}：对于 G' 中形如 $A \to BC$ 的每个产生式，我们将 C 存入 T_{suf}，并将 $\mathrm{rev}(B)$ 存入 T_{pre}。对于形如 $A \to B^k$ 的每个产生式，我们将 B、B^2、B^{k-2} 和 B^{k-1} 存入 T_{suf}，并将这些字符串的反转字符串存入 T_{pre}。对于 $j \in \{1,2\}$ 和 $s \in \mathrm{Splits}'(G', P_j)$，定义 $S_j(s) = \mathrm{rev}(U)V$，其中 U 是 $\mathrm{rev}(P_j[\ldots s])$ 在 T_{pre} 中所在位置的标签，而 V 是 $P_j(s\ldots]$ 在 T_{suf} 中所在位置的标签。令 $l_j(s) = |\mathrm{rev}(U)|$，且 $\Delta_j(s) = l_j(s) - s$。

考虑一个非终结符 A，其展开 A 包含 P_1, P_2的一个相关共现 (q_1, q_2)。

命题 4.3. 存在 $s \in \mathrm{Splits}'(G', P_2)$，使得 $p_2 = q_2 - \Delta_2(s)$ 是 $S_2(s)$在 A 中的一个出现位置，且 $[p_2, p_2 + |S_2(s)|) \supseteq [q_2, q_2 + |P_2|)$。

证明. 下面证明，存在 A 的一个后代 A' 以及一个分割点 $s \in \mathrm{Splits}'(G', P_2)$，使得以下两种情况之一成立：$\mathrm{rev}(P_2[\ldots s])$ 是 $\mathrm{rev}(\mathrm{head}(A'))$ 的前缀，且 $P_2(s\ldots]$ 是 $\mathrm{tail}(A')$ 的前缀；或者 A' 与规则 $A' \to (B')^k$ 相关联，$\mathrm{rev}(P_2[\ldots s])$ 是 $\mathrm{rev}((B')^2)$ 的前缀，且 $P_2(s\ldots]$ 是 $(B')^{k-2}$ 的前缀。根据 T_{pre}、T_{suf} 和 $S_2(s)$ 的定义，命题得证。

如果 q_2 在 A中是相关的，则存在一个分割点 $s \in \mathrm{Splits}'(G', P_2)$，使得 $\mathrm{rev}(P_2[\ldots s])$ 是 $\mathrm{rev}(\mathrm{head}(A))$ 的一个前缀，并且根据定义，$P_2(s\ldots]$ 是 $\mathrm{tail}(A)$ 的一个前缀。如果 q_2 不相关，则根据共现的定义，$q_2 \geq |\mathrm{head}(A)|$。根据断言 2.5，存在 A' 的一个后代 A，对应于一个子串 $A[\ell\ldots r]$，其中要么 $(q_2 - \ell)$ 是相关的（于是我们可以重复上述论证），要么 A' 与某条规则 $A' \to (B')^k$ 相关联，且对于某个 $0 \leq k' \leq k$，$(q_2 - \ell) - k' \cdot |B'|$是相关的。考虑后一种情况。如果 $A' = A$，则 $k' = 1$，否则$q_1 < q_2' = q_2 - |B'| < q_2$ 是在 A 中 P_2 的一个出现位置，这与共现的定义相矛盾（回忆一下，(q_1, q_2) 是一个相关共现，因此根据定义 $q_1 < |\mathrm{head}(A)|$），从而 $s = |(B')^2| - q_2 + \ell \in \mathrm{Splits}'(G', P_2)$，$\mathrm{rev}(P_2[\ldots s])$ 是 $\mathrm{rev}((B')^2)$ 的一个前缀，并且$P_2(s\ldots]$ 是 $(B')^{k-2}$ 的一个前缀。如果 $A' \neq A$，则类似地可得 $k' = 0$，这意味着 $s = |B'| - q_2 + \ell \in \mathrm{Splits}'(G', P_2)$，$\mathrm{rev}(P_2[\ldots s])$ 是 $\mathrm{rev}(B')$ 的一个前缀，并且 $P_2(s\ldots]$ 是 $(B')^{k-1}$ 的一个前缀。 □

由于共现的定义并不对称，q_1 不具备相同的性质。不过，可以证明一个类似的结论：

引理 4.4. 存在 $s \in \mathrm{Splits}'(G', P_1)$ 以及 $S_1(s)$ 中的一个出现位置 p_1，使得$[p_1, p_1 + |S_1(s)|) \supseteq [q_1, q_1 + |P_1|)$，并且以下条件至少有一项成立：

1. $q_1 - \Delta_1(s)$ 是 $S_1(s)$的一个出现位置；
2. q_2 是A中 P_2 的相关出现，$S_1(s)$ 的周期等于 P_1的周期，并且存在一个整数 k，使得 $p_1 = q_1 - \Delta_1(s) - \pi_1 \cdot k$且 $q_2 + \pi_1 - 1 \leq p_1 + |S_1(s)| - 1 \leq q_2 + |P_2| - 1$。

证明。 如果 A 中带有分割点 $s \in \text{Splits}'(G', P_1)$的 P_1 的相关出现是 q_1，那么rev($P_1[\ldots s]$)是rev(head(A)) 的前缀，并且 $P_1(s\ldots]$ 是tail(A)的前缀，因此根据 T_{pre} 和 T_{suf}的定义，第一种情况成立。

否则，根据断言 2.5，存在一个与子串$A[\ell\ldots r]$ 相对应的头部(A)的后代 A'，使得要么 $(q_1 - \ell)$ 是相关出现（此时我们可以重复上述论证），要么 A' 与规则 $A' \to (B')^k$ 相关，并且对于某个 $0 \le k' \le k$, $(q_1 - \ell) - k' \cdot |B'|$ 是 A' 中 P_1 的一个相关出现，其分割点为 $s \in \text{Splits}'(G', P_1)$。考虑后一种情况。我们必有 (1) $q_1 + |P_1| - 1 + |B'| \ge r$ 或 (2) $q_1 + |B'| - 1 \ge q_2$，因为如果这两个不等式都不成立，那么 $q_1 < q_1 + |B'| \le q_2$ 就是 A 中 P_1 的一个出现位置，这与共现的定义相矛盾。此外，如果 (1) 成立，则根据定义，存在一个分割点$s' \in \text{Splits}'(G', P_1)$（可能不同于上面的分割点 s），使得 rev($P_1[\ldots s']$) 是 rev($(B')^{r-1}$) 的前缀，并且 $P_1(s'\ldots]$ 是 B' 的前缀，于是我们落入引理的第一种情况。

从现在起，假设 (2) 成立而 (1) 不成立。由于 $q_1 + |B'| \le r \le |\text{head}(A)|$ and (q_1, q_2) 是相关共现，q_2 必定是 A 中 P_2 的相关出现位置。若 $|P_1| - s \le |(B')^2|$，则 rev($P_1[\ldots s]$) 是 rev(B') 的前缀，且 $P_1(s\ldots]$ 是 $(B')^2$ 的前缀，因此$q_1 - \Delta_1(s)$ 是 $S_1(s)$ 的一个出现位置。否则，由 Fine–Wilf 周期性引理，A'、P_1 和 $S_1(s)$ 的周期相同，因为 P_1，因此 $S_1(s)$ 至少跨越 A' 的两个周期。根据周期性，$S_1(s)$ 在 A 的位置 $q_1 - \Delta_1(s) - |B'| \cdot k$ 处出现。令 p_1 为这些位置中满足 $p_1 + |S_1(s)| - 1 \ge q_1 + |P_1| - 1$ 的最左位置。由于 (1) 不成立，该位置定义良好；此外，$[q_1, q_1 + |P_1|) \subseteq [p_1, p_1 + |S_1(s)|)$，因为 $s \le l_1(s)$ 且$|S_1(s)| - l_1(s) \ge |P_1| - s$。对于某个整数 k''，有 $p_1 = q_1 - \Delta_1(s) - \pi_1 \cdot k''$（因为$|B'|$ 是 π_1的倍数）；并且 $q_2 + \pi_1 - 1 \le q_1 + 2|B'| - 1 \le q_1 + |P_1| - 1 \le p_1 + |S_1(s)| - 1 \le r < q_2 + |P_2| - 1$，最后一个不等式成立是因为 q_2 是 A 中的相关出现位置。引理的结论得证。

□

我们总结命题 4.3 和引理 4.4：

推论 *4.5.* 设 (q_1, q_2) 是非终结符 A 的展开中 P_1, P_2 的一次共现。存在分割点 $s_1 \in \text{Splits}'(G', P_1)$ $s_2 \in \text{Splits}'(G', P_2)$，以及 $S_1(s_1)$ 的出现位置 p_1 和$S_2(s_2)$ 的出现位置 p_2，其中 $[p_1, p_1 + |S_1(s_1)|) \supseteq [q_1, q_1 + |P_1|)$ 且 $[p_2, p_2 + |S_2(s_2)|) \supseteq [q_2, q_2 + |P_2|)$，使得以下条件至少有一项成立：

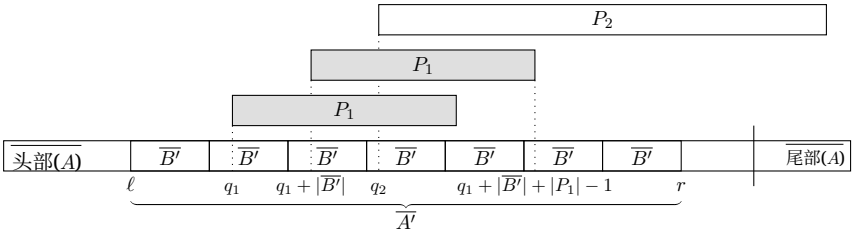
1. 出现位置 p_1 要么是相关的，要么 $p_1 + |S_1(s_1)| - 1 \le |\text{head}(A)|$。出现位置 p_2 要么是相关的，要么 $p_2 > |\text{head}(A)|$。此外，$p_1 = q_1 - \Delta_1(s_1)$ 且 $p_2 = q_2 - \Delta_2(s_2)$。 *2.* 出现位置 p_2 是相关的，且 $p_1 \le |\text{head}(A)|$。此外，$p_2 = q_2 - \Delta_2(s_2)$，$S_1(s)$ 的周期等于 P_1 的周期，并且存在一个整数 k，使得 $p_1 = q_1 - \Delta_1(s_1) - \pi_1 \cdot k$ 且 $p_2 + \pi_1 - 1 \le p_1 + |S_1(s_1)| - 1 \le p_2 + |S_2(s_2)| - 1$。

反转后的观察同样成立：

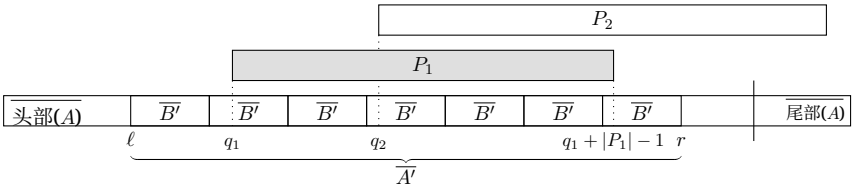
观察 4.6. 若 p_j 是 A 中 $S_j(s)$ 的一个出现位置，则 $q_j = p_j + \Delta_j(s)$ 是 P_j 的一个出现位置。此外，若 $S_1(s)$ 的周期为 π_1，则 $q_1 + \pi_1 \cdot k$、$0 \le k \le \lfloor(|S_1(s)| - q_1 - |P_1|)/\pi_1\rfloor$ 是 A 中 P_1 的出现位置。

最后，以下这个显而易见的观察对于给查询算法的时间复杂度设定上界非常重要：

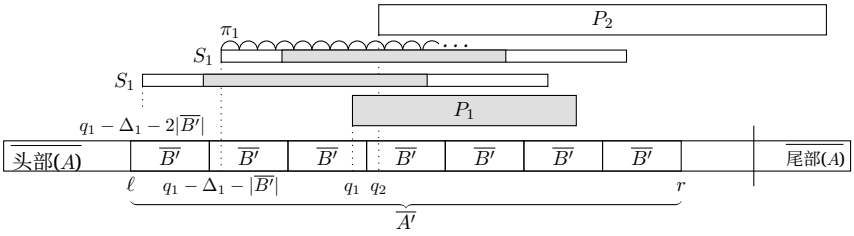
观察 4.7。 如果一个字符串包含一对出现位置 (q_1, q_2)，分别对应 P_1 和 P_2，且满足 $0 \le q_2 - q_1 \le b$，那么它包含一个 b-近距离共现，其中涉及 P_1 和 P_2。



(a) 如果 (1) 和 (2) 均不成立，那么 (q_1，q_2) 不连续。



(b) (1) 成立，而 (2) 不成立。



(c) (2) 成立，(1) 不成立，并且 $|P_1|-s\geq\overline{(B')^2}$。

图 2：引理 4.4 的子情况。

4.2 索引

索引的第一部分是定理 3.1 的数据结构，以及 Christiansen 等人提出的索引 [8]:

事实 4.8 ([8, 引言和定理 6。12])。存在一种占用 $O(g$ 对数$^2\, N)$空间的数据结构，可在 $O(m+\mathrm{occ})$时间内找到模式 $P[1\ldots m]$ 在 S 中的 Oocc 个出现位置。

索引的第二部分是字典树 T_{pre} 和 T_{suf}，并按下文所述进行扩充。考虑四元组 (u_1, u_2, v_1, v_2)，其中 u_1 和 u_2 是 T_{pre} 中的节点，v_1 和 v_2 是 T_{suf}中的节点。令 U_1, U_2, V_1, V_2 分别为 u_1, u_2, v_1, v_2的标签。定义 $S_1 = \mathsf{rev}(U_1)V_1$ 和$S_2 = \mathsf{rev}(U_2)V_2$，并令 $l_1 = |\mathsf{rev}(U_1)|$ 和 $l_2 = |\mathsf{rev}(U_2)|$。

首先，我们存储一棵二叉搜索树 $\mathcal{T}_1(u_1, u_2, v_1, v_2)$，对于每个非终结符 $\underline{A}$，其中至多包含六个整数 $d = p_2 - p_1$，其中 p_1, p_2 是 S_1, S_2 在 A中的出现位置，并满足以下至少一个条件:

1. p_1 是满足 $p_1 + |S_1| - 1 < |\mathsf{head}(A)|$ 的 $\underline{S_1 \text{ 最右}}$出现位置，且 p_2 $\overline{\text{是满足}}$ $p_2 \geq |\mathsf{head}(A)|$ 的 S_2 最左出现位置；2. p_1 是 S_1 的一个相关出现，其分割点为 l_1，且 p_2 是满足 $p_2 \geq |\mathsf{head}(A)|$的 S_2最左出现位置；3. p_1 是 S_1 的一个相关出现，其分割点为 l_1，p_2 是 S_2 的一个相关出现，其分割点为 l_2;

4. p_2 是 S_2 的一个相关出现，其分割点为 l_2，且 p_1 是满足 $p_1 + |S_1| - 1 < p_2$的 S_1最右出现位置；

5. p_2 是 S_2 的一个相关出现，其分割点为 l_2，且 p_1 是 head(A) 中 S_1 的最左或第二最左出现位置，使得 $p_1 < p_2 \le p_1 + |S_1| - 1 < p_2 + |S_2| - 1$。

其次，我们存储一个非终结符列表 $\mathcal{L}(u_2, v_2)$，其展开包含 S_2 的一个相关出现，分割点为 l_2。此外，对于每个 $k \in [0, \log N]$，若有定义，我们存储：

1. S_1 在 S_2 中满足 $p_1 + (|S_1| - 1) \le l_2 - 2^k$ 的最右出现位置 p_1；

2. S_1 在 S_2 中满足 $p_1' \le l_2 - 2^k \le p_1' + |S_1| - 1$ 的最左出现位置 p_1'；

3. 满足 $p_1'' \le l_2 - 2^k \le p_1'' + |S_1| - 1$ 的 S_2 中 S_1 的最右出现位置 p_1''。

最后，我们计算并记忆 S_1 的周期 π_1。如果该周期定义良好（即 S_1 是周期性的），我们构建一棵二叉搜索树 $\mathcal{T}_2(u_1, u_2, v_1, v_2)$。考虑一个非终结符 A，其中包含 S_2 的一个相关出现位置 p_2，并带有一个分割点 l_2。令 p_1 为满足$p_1 \le p_2 \le p_1 + |S_1| - 1 \le p_2 + |S_2| - 1$ 的 S_1 的最左出现位置，并令 p_1' 为最右出现位置。如果 p_1 和 p_1' 存在（p_1 可能等于 p_1'），且 $p_1' + |S_1| - 1 \ge p_2 + \pi_1 - 1$，我们就向树中添加一个整数 $(p_1' - p_1)/\pi_1$，并将其与 A 关联。我们还记忆一个数 $\mathrm{ov}(S_1, S_2) = p_2 - p_1'$；根据推论 2.1，它不依赖于 A，因此定义良好（它对应于 S_2 的最长前缀，该前缀具有周期 π_1）。

断言 4.9。该数据结构占用 $O(g^5 \log^5 N)$ 空间。

证明。定理 3.1 的数据结构占用 $O(g^2 \log^4 N)$ 空间。Christiansen 等人的索引占用 $O(g \log^2 N)$ 空间。根据引理 2.8，字典树占用 $O(g') = O(g \log N)$ 空间。共有 $O((g')^4)$ 个四元组 (u_1, u_2, v_1, v_2)，每个四元组对应的树占用 $O(g')$ 空间。S_1 在 S_2 中的出现位置数组占用 $O(\log N)$ 空间。因此，该数据结构总共占用 $O(g^5 \log^5 N)$ 空间。 □

4.3 查询

回顾一下，查询由两个字符串 P_1, P_2 组成，每个字符串的长度至多为 m，以及一个整数 b；我们必须找出 P_1, P_2 在 S中的所有 b-近距离共现，设其数量为 occ。

我们首先使用线性时间、常数空间的模式匹配算法（例如 [11]），检查 P_2 是否出现在 P_1 中。如果是，令 q_2 为首次出现的位置。如果$q_2 > b$，那么 P_1, P_2 在 S中不存在距离为 b 的近距离共现。否则，要找出 P_1, P_2 在 S 中的所有距离为 b 的近距离共现（这些共现始终由 S 中 P_1 的一次出现位置和 P_1中 P_2 的首次出现位置组成），只需找出 P_1 在 S中的所有出现位置；我们使用 Christiansen 等人的索引，在时间 $O(|P_1| + \mathrm{occ}) = O(m + \mathrm{occ})$ 内完成此操作。

从现在起，假设 P_2 不是 P_1的子串。令 $\mathcal{N}$ 为 G' 中所有非终结符构成的集合，这些非终结符的展开包含 P_1, P_2的一次相关的距离为 b 的近距离共现。根据命题 4.2，$|\mathcal{N}| \le \mathrm{occ}$。

引理 4.10. 假设 P_2 不是 P_1 的子串。可以在 $O(m + (1 + \mathrm{occ}) \log^3 N)$ 时间内检索一个集合 $\mathcal{N}' \supset \mathcal{N}$, $|\mathcal{N}'| = O(\mathrm{occ} \log N)$。

证明。我们首先通过引理 2.7，在 $O((|P_1| + |P_2|) \log N) = O(m \log N)$ 时间内计算 $\mathrm{Splits}'(G', P_1)$ 和 $\mathrm{Splits}'(G', P_2)$（或者提供证书，证明 P_1 或 P_2 未在 S中出现；在这种情况下，P_1、P_2 在 S 中没有共同出现的位置，问题便解决了）。回顾一下，$|\mathrm{Splits}'(G', P_1)|$、$|\mathrm{Splits}'(G', P_2)| \in O(\log N)$。对于每一对固定的分割点 $s_1 \in \mathrm{Splits}'(G', P_1)$,

$s_2 \in \text{Splits}'(G', P_2)$ 和 $j \in \{1, 2\}$，我们计算 T_{pre} 中以 $\text{rev}(P_j[\ldots s_j])$ 为前缀的字符串区间；该区间对应于 T_{pre} 中 $\text{rev}(P_j[\ldots s_j])$ 的位置 u_j，以及 T_{suf} 中以 $P_j(s_j \ldots]$ 为前缀的字符串区间；该区间对应于 T_{suf} 中 $P_j(s_j \ldots]$ 的位置 v_j。根据引理 2.8，计算所有分割点对应的区间需要 $O(m+\log^2 N)$ 时间。考虑字符串 $S_1 = \text{rev}(U_1)V_1$ 和 $S_2 = \text{rev}(U_2)V_2$，其中 U_1、U_2、V_1、V_2 分别是 u_1、v_1、u_2、v_2的标签。令 $l_1 = |\text{rev}(U_1)|$、$\Delta_1 = l_1 - s_1$、$l_2 = |\text{rev}(U_2)|$、$\Delta_2 = l_2 - s_2$，以及 $\Delta = \Delta_1 - \Delta_2$。

考虑非终结符 A 的展开中 P_1，P_2 的一个相关共现 (q_1, q_2)。根据推论 4.5，q_1，q_2 蕴含存在 S_1，S_2 的出现位置 p_1，p_2，使得 $[p_1, p_1 + |S_1|) \supseteq [q_1, q_1 + |P_1|)$ 且 $[p_2, p_2 + |S_2|) \supseteq [q_2, q_2 + |P_2|)$。我们的索引必须处理推论 4.5 的两种情况。我们考虑图 3 中定义的八个子情况，它们描述了 p_1 和 p_2 的所有可能位置。

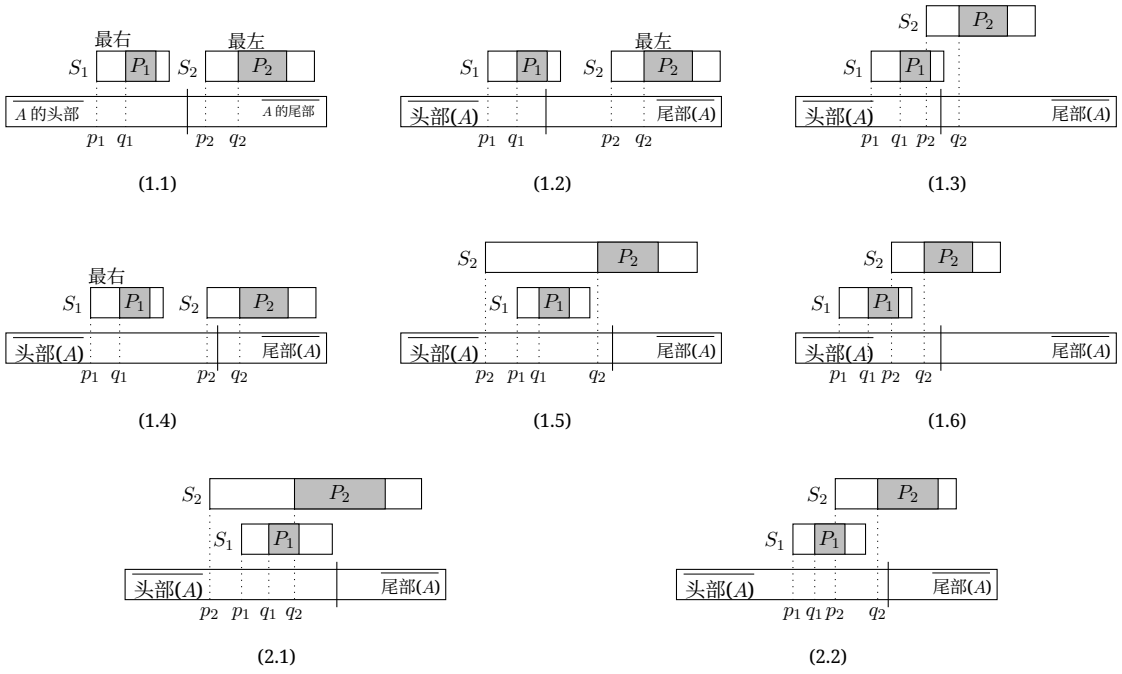


图3：假设 S_1 不包含 S_2。该图展示了 A 中 S_1, S_2 的出现位置 p_1, p_2 的所有可能位置。在推论 4.5 的情况 1 中，有六个子情况：(1.1)$p_1 + |S_1| - 1 \le |\text{头部}(A)|$, $p_2 > |\text{头部}(A)|$；(1.2) p_1 是 S_1, $p_2 > |\text{头部}(A)|$ 的相关出现位置；(1.3) p_1, p_2 是相关的；(1.4) p_2 是相关的，$p_1 + |S_1| - 1 \le p_2$；(1.5) p_2 是相关的，$p_2 < p_1 \le p_1 + |S_1| - 1 \le p_2 + |S_2| - 1$；(1.6) p_2 是相关的，$p_1 < p_2 < p_1 + |S_1| - 1 \le p_2 + |S_2| - 1$。根据共现的定义以及观察 4.6，在子情况 (1.1) 和 (1.4) 中，p_1 必须尽可能靠右；在子情况 (1.1) 和 (1.2) 中，p_2 必须尽可能靠左。在情况 2 中，有两个子情况：(2.1) p_2 是相关的且 $p_2 \le p_1 \le p_1 + |S_1| - 1 \le p_2 + |S_2| - 1$；(2.2) p_2 是相关的且 $p_1 < p_2 < p_2 + \pi_1 - 1 \le p_1 + |S_1| - 1$，其中 π_1 是 S_1 的周期。在所有子情况中，$q_2 = p_2 + \Delta_2$。在子情况 (1.1)-(1.6) 中，$q_1 = p_1 + \Delta_1$；在子情况 (2.1) 和 (2.2) 中，对于某个整数 k，$q_1 = p_1 + \Delta_1 + k \cdot \pi_1$。

子情况 (1.1)-(1.4)。为检索非终结符，我们查询 $\mathcal{T}_1(u_1, u_2, v_1, v_2)$，以找出属于范围 $[\Delta, \Delta + b]$ 的所有整数（以及相应的非终结符）。回想一下，对于每个非终结符 A，树中存储一个整数 $d = p_2 - p_1$，其中 p_1 是 A 中 S_1 的某个出现位置的起始位置，也是 S_2 中 p_2 的某个出现位置的起始位置。根据观察 4.6，$p_1 + \Delta_1$ 是 P_1 的一个出现位置，$p_2 + \Delta_2$ 是 P_2 的一个出现位置。它们之间的距离属于 $[0, b]$ 当且仅当 $d \in [\Delta, \Delta + b]$。根据观察 4.7，每个检索到的非终结符都包含一对近距离共现 (q_1, q_2)。另一方面，如果 A 包含一对与子情况 (1.1)-(1.4) 之一相对应的共现 (q_1, q_2)，那么

根据推论 4.5，$p_1 = q_1 - \Delta_1$ 是 S_1 的一个出现位置，$p_2 = q_2 - \Delta_2$ 是 S_2 的一个出现位置；并且根据构造，$\mathcal{T}_1$(u_1，u_2，v_1，v_2) 存储一个整数 $d = p_2 - p_1$。因此，该查询检索到与子情况 (1.1)-(1.4) 相对应的所有非终结符。

子情况 (1.5) 和 (2.1)。我们必须判断 S_2 中的 P_1 出现位置是否与 S_2 中的 P_2 出现位置 Δ_2 形成 b- 近距离共现；如果是，则报告所有其展开包含 S_2 与分割点 l_2 的相关共现的非终结符，这些非终结符恰好存储在列表 $\mathcal{L}(u_2, v_2)$ 中。令 $k = \lceil \log(s_2) \rceil$。回顾一下，索引为 k 存储以下信息：

1. p_1，S_2 中满足 $p_1 + (|S_1| - 1) \leq l_2 - 2^k$ 的 S_1 的最右出现位置；2. p'_1，S_2 中满足 $p'_1 \leq l_2 - 2^k \leq p_1 + (|S_1| - 1)$ 的 S_1 的最左出现位置；3. p''_1，S_2 中满足 $p''_1 \leq l_2 - 2^k \leq p''_1 + (|S_1| - 1)$ 的 S_1 的最右出现位置。

（见图 4）。根据观察 4.6，p_1 的 S_1 出现位置会诱导出P_1的一个出现位置 $q_1 = p_1 + \Delta_1$。此外，如果 S_1是周期为 π_1的周期性模式，那么 $q_1 + \pi_1 \cdot k$、$0 \leq k \leq \lfloor (|S_1| - q_1 - |P_1|) / \pi_1 \rfloor$也都是 P_1的出现位置。我们可以在常数时间内判断这些出现位置中任意一个与q_2之间的距离是否属于 $[0, b]$；如果是，那么根据观察 4.7，S_2中就包含 P_1和 P_2的一个 b近距离共现。其次，根据推论 2.1，如果 S_1 不是周期性的，那么 p'_1 与 p''_1 之间不存在 S_1 的出现位置，而 p'_1、p''_1 根据观察 4.6 会诱导出 P_1的出现位置 $p'_1 + \Delta_1$、$p''_1 + \Delta_1$。否则，在每个位置 $p'_1 + \Delta_1 + k \cdot \pi_1$、$0 \leq k \leq \lfloor (|S_1| + p''_1 - |P_1| - p'_1) / \pi_1 \rfloor$处都存在 P_1 的出现位置。同样，我们可以在常数时间内判断其中任意一个位置与 S_2 中 P_2 的出现位置 Δ_2 之间的距离是否属于 $[0, b]$。最后，令 q_1 为区间 $[l_2 - 2^k + 1, \Delta_2]$内 S_2 中 P_1 的最右出现位置。我们通过事实 A.4 提取 S_2（$l_2 - 2^k$，$\Delta_2 + |P_2|$），并使用针对 P_1的线性时间模式匹配算法搜索 q_1，耗时$O(|P_1| + |P_2|) = O(m)$。如果 $0 \leq \Delta_2 - q_1 \leq b$，那么 S_2中就存在 P_1和 P_2的一个 b近距离共现。正确性由推论 4.5、观察 4.6 和观察 4.7 得证。

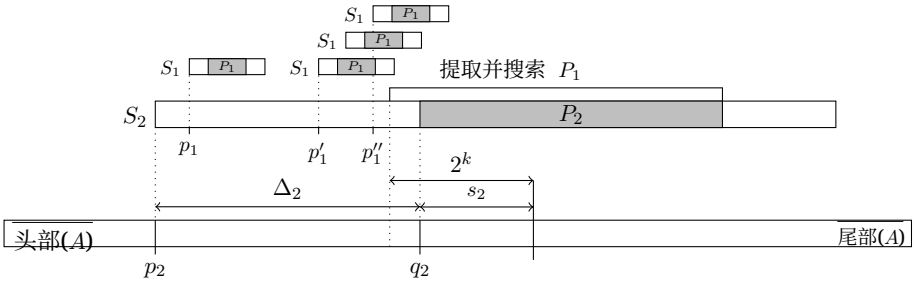


图 4：子情况 (1.5) 和 (2.1) 的查询算法。

子情况 (2.2)。令 π_1 为 S_1的周期。我们检索与整数 $q \in \mathcal{T}_2(u_1, u_2, v_1, v_2)$ 相关联的非终结符，使得区间 $I = [a, b]$ 和 $[\ell, q]$ 的交集非空，其中

$$a = \lceil (\Delta - \mathrm{ov}(S_1, S_2))/\pi_1 \rceil,\ b = \lfloor (\Delta - \mathrm{ov}(S_1, S_2) + b)/\pi_1 \rfloor \text{ and } \ell = -\lfloor (|S_1| - |P_1| - \Delta_1)/\pi_1 \rfloor$$

（索引的描述中给出了 $\mathrm{ov}(S_1, S_2)$ 的定义。）由于 ℓ 固定，我们至多通过一次二叉树搜索即可实现该查询：如果 $b \leq \ell$，则输出为空；如果 $a \leq \ell \leq b$，则必须输出所有整数；如果 $\ell \leq a$，则必须输出所有 $q \geq b$。现在来说明该算法为何正确。考虑一个非终结符 A，其中 $\mathcal{T}_2(u_1, u_2, v_1, v_2)$ 存储了一个整数 q。根据构造，A 包含 S_2 的一个相关出现位置，其分割点为 l_2。位置 $p_1 = |\mathrm{head}(A)| - l_2 - \mathrm{ov}(S_1, S_2) - q \cdot \pi_1$ 是 A 中 S_1 的最左出现位置，满足 $p_1 \leq$

$p_2 \le p_1 + |S_1| - 1$，而 $p_2 = |\mathsf{head}(A)| - l_2 - \mathsf{ov}(S_1, S_2)$ 是最右出现位置。因此，对于每个 $-\lfloor(|S_1| - |P_1| - \Delta_1)/\pi_1\rfloor \le q' \le q$，都存在 P_1 的一个出现位置 $q_1 = |\mathsf{head}(A)| - l_2 - \mathsf{ov}(S_1, S_2) - q' \cdot \pi_1 + \Delta_1$。$S_2$ 的出现位置意味着 $A\mathsf{head}(|-s_2|)\ P_2$ 是 $q_2 = |$ 的一个出现位置。我们有 $0 \le q_2 - q_1 = q' \cdot \pi_1 + \mathsf{ov}(S_1, S_2) - \Delta \le b$ 当且仅当 $\Delta - \mathsf{ov}(S_1, S_2) \le q' \cdot \pi_1 \le \Delta - \mathsf{ov}(S_1, S_2) + b$，这等价于 $[\ell, q] \cap I \ne \emptyset$。由此可知，我们检索到了与子情形（2.2）对应的每个非终结符。另一方面，根据观察 4.7，每个检索到的非终结符的展开都包含 P_1 和 P_2 的一个 b 近距离共现。

子情况（1.6）。我们说明，与此子情况对应的所有非终结符都已报告完毕，无事可做。考虑一个非终结符 A，其展开中包含 S_2 的一个相关出现位置 p_2。如果满足 $p_1 \le p_2 \le p_1 + |S_1| - 1 \le p_2 + |S_2| - 1$ 的 S_1 出现位置 p_1 至多有两个，我们会在查询 $\mathcal{T}_1(u_1, u_2, v_1, v_2)$ 时处理它们（子情况（1.1）至（1.4））。否则，根据推论 2.1，S_1 是周期性的，并且存在一个 S_1 的出现位置 p_1'，使得 $p_1' \le p_2 < p_2 + \pi_1 \le p_1 + |S_1| - 1 < p_2 + |S_2| - 1$。与此情况对应的非终结符会在查询 $\mathcal{T}_2(u_1, u_2, v_1, v_2)$ 时报告（子情况（2.2））。

时间复杂度。如上所示，该算法报告一个包含 $\mathcal{N}' \supset \mathcal{N}$ 个非终结符的集合，且 $\mathcal{N}'$ 中的每个非终结符都包含一个 b-近距离共现。根据命题 4.2，且由于 G' 的高度为 $h = O(\log N)$，我们有 $|\mathcal{N}'| = O(\mathsf{occ} \log N)$。此外，对于 P_1, P_2 的一对固定分割点，$\mathcal{N}'$ 中的每个非终结符都至多被报告常数次。由于 $|\mathsf{Splits}'(G', P_1)| \cdot |\mathsf{Splits}'(G', P_2)| = O(\log^2 N)$，输出的总大小为 $|\mathcal{N}'| \cdot O(\log^2 N) = O(\mathsf{occ} \cdot \log^3 N)$。因此，我们得到该算法的运行时间为 $O(m + \log^3 N + \mathsf{occ} \log^3 N) = O(m + (1 + \mathsf{occ}) \log^3 N)$，如所需。 □

利用定理 3.1 检索到集合 $\mathcal{N}'$ 后，我们可以为 $\mathcal{N}'$ 中的每个非终结符找出所有与 b 接近的相关共现。实际上，我们的算法采用朴素方法，先计算 $\mathcal{N}'$ 中某个非终结符的所有相关共现，再选出其中与 b 接近的共现。通过分类讨论可以证明，对于非终结符 A，一个相关共现总是由 P_2 的一个出现位置和 P_1 的一个先前出现位置组成；其中，~~P_2~~ 的该出现位置要么是相关的，要么是 $\mathsf{tail}(A)$ 中最左的出现位置。直观而言，这使我们能够高效地计算所有相关共现，并保证其数量较少。形式化地，我们证明以下结论：

引理 4.11. 假设 P_2 不是 P_1 的子串。经过 $O(m \log N + \log^2 N)$ 时间的预处理后，定理 *3.1* 的数据结构能够在 $O(\log^3 N \log\log N)$ 时间内，计算给定非终结符 A 的展开中 P_1 的所有 b 接近的相关共现，P_2。

Christiansen 等人的索引的一部分 [8] 是对 G'解析树进行剪枝后得到的副本。他们展示了如何遍历该树，在已知模式在各非终结符中的相关出现位置时，报告该模式的所有出现位置。通过基本上使用相同的算法，我们可以在每次共现的摊销常数时间内报告所有 b-近距离共现，由此完成定理 1.1 的证明。（为完整起见，我们在附录 C 的引理 C.2 中给出这一步骤的全部细节。）

参考文献

[1] Amir Abboud、Arturs Backurs、Karl Bringmann 和 Marvin Künnemann。分析压缩数据的细粒度复杂性：量化相较于解压后求解的改进。载于第 *58* 届 *FOCS* 论文集，第 192–203 页，2017 年。[2] Amir Abboud、Arturs Backurs、Karl Bringmann 和 Marvin Künnemann。文法压缩线性代数的不可能性结果。载于第 *34* 届 *NeurIPS* 论文集，第 8810–8823 页，2020 年。[3] Djamal Belazzougui、Paolo Boldi、Rasmus Pagh 和 Sebastiano Vigna。小空间快速前缀搜索及其应用。载于第 *18* 届 *ESA* 论文集，第 427–438 页，2010 年。[4] Philip Bille、Inge Li Gørtz、Max Rishøj Pedersen 和 Teresa Anna Steiner。连续出现的间隔索引。载于第 *32* 届 *CPM* 论文集，第 10:1–10:19 页，2021 年。[5] Philip Bille、Gad M. Landau、Rajeev Raman、Kunihiko Sadakane、Srinivasa Rao Satti 和 Oren Weimann。文法压缩字符串和树的随机访问。*SIAM J.Comput.*，44(3):513–539，2015 年。[6] Timothy M. Chan。字 RAM 上的持久化前驱搜索与正交点定位。*ACM Trans. Algorithms*，9(3):22:1–22:22，2013 年。[7]Moses Charikar、Eric Lehman、Ding Liu、Rina Panigrahy、Manoj Prabhakaran、April Rasala、Amit Sahai 和 Abhi Shelat。逼近最小文法：自然模型中的柯尔莫哥洛夫复杂性。载于第 *34* 届 *STOC* 论文集，第 792–801 页，2002 年。[8] Anders Roy Christiansen、Mikko Berggren Ettienne、Tomasz Kociumaka、Gonzalo Navarro 和 Nicola Prezza。最优时间字典压缩索引。*ACM Trans. Algorithms*，17(1):8:1–8:39，2021 年。[9] Francisco Claude 和 Gonzalo Navarro。改进的基于文法的压缩索引。载于第 *19* 届 *SPIRE* 论文集，第 180–192 页，2012 年。[10] Francisco Claude、Gonzalo Navarro 和 Alejandro Pacheco。具有对数搜索时间的文法压缩索引。*J. Comput. Syst. Sci.*，118:53–74，2021 年。[11] Maxime Crochemore。常数空间字符串匹配。载于第 *8* 届 *FSTTCS* 论文集，第 80–87 页，1988 年。[12] Diego Díaz-Domínguez、Gonzalo Navarro 和 Alejandro Pacheco。具有局部一致性性质的基于LMS的文法自索引。载于第 *28* 届 *SPIRE* 论文集，2021 年。[13] Paolo Ferragina 和 Rossano Venturini。压缩文本的索引。载于《数据库系统百科全书》（第 *2* 版）。Springer，2018 年。[14] Nathan J. Fine 和 Herbert S. Wilf。周期函数的唯一性定理。*Proc.Am. Math. Soc.*，16(1):109–114，1965 年。[15] Johannes Fischer、Travis Gagie、Tsvi Kopelowitz、Moshe Lewenstein、Veli Mäkinen、Leena Salmela 和 Niko Välimäki。禁用模式。载于第 *10* 届 *LATIN* 论文集，第 327–337 页，2012 年。[16] Travis Gagie、Pawel Gawrychowski、Juha Kärkkäinen、Yakov Nekrich 和 Simon J. Puglisi。基于LZ77的自索引及更快的模式匹配。载于第 *11* 届 *LATIN* 论文集，第 731–742 页，2014 年。

[17] Travis Gagie、Gonzalo Navarro 和 Nicola Prezza。BWT-run 有界空间内的最优时间文本索引。载于第 *29* 届 *SODA* 论文集，第 1459–1477 页，2018 年。[18]Pawel Gawrychowski、Adam Karczmarz、Tomasz Kociumaka、Jakub Lacki 和 Piotr Sankowski。最优动态字符串。载于第 *29* 届 *SODA* 论文集，第 1509–1528 页，2018 年。[19] Daniel Gibney 和 Sharma V. Thankachan。用于正则表达式匹配的文本索引。*Algorithms*，14(5):133，2021 年。[20] Leszek Gąsieniec、Roman M. Kolpakov、Igor Potapov 和 Paul Sant。基于文法压缩文件的实时遍历。载于第 *15* 届 *DCC* 论文集，第 458 页，2005 年。[21] Dan Gusfield。字符串、树和序列算法——计算机科学与计算生物学。剑桥大学出版社，1997 年。[22] Wing-Kai Hon、Rahul Shah、Sharma V. Thankachan 和 Jeffrey Scott Vitter。多模式查询的字符串检索。载于第 *17* 届 *SPIRE* 论文集，第 55–66 页，2010 年。[23] Wing-Kai Hon、Rahul Shah、Sharma V. Thankachan 和 Jeffrey Scott Vitter。针对含排除模式的查询进行文档列举。载于第 *23* 届 *CPM* 论文集，第 185–195 页，2012 年。[24] Richard M Karp 和 Michael O Rabin。高效的随机化模式匹配算法。*IBM J. Res. Dev*，31(2):249–260，1987 年。[25] John C. Kieffer 和 En-Hui Yang。基于文法的编码：一类新的通用无损源编码。*IEEE Trans. Inf. Theory*，46(3):737–754，2000 年。[26] Tsvi Kopelowitz 和 Robert Krauthgamer。颜色距离预言机与片段。载于第 *27* 届 *CPM* 论文集，第 24:1–24:10 页，2016 年。[27] Tsvi Kopelowitz、Seth Pettie 和 Ely Porat。由 3SUM 猜想导出的更高下界。载于第 *27* 届 *SODA* 论文集，第 1272–1287 页，2016 年。[28] Kasper Green Larsen、J. Ian Munro、Jesper Sindahl Nielsen 和 Sharma V. Thankachan。若干字符串索引问题的难度。*Theor. Comput. Sci.*，582:74–82，2015 年。[29] Moshe Lewenstein。文本索引中的正交范围查询。载于空间高效数据结构、流与算法，第 267–302 页，2013 年。[30] Veli Mäkinen 和 Gonzalo Navarro。压缩文本索引。载于算法百科全书，第 394–397 页，2016 年。[31] Udi Manber 和 Eugene W. Myers。后缀数组：一种在线字符串搜索的新方法。*SIAM J. Comput.*，22(5):935–948，1993 年。[32] S. Muthukrishnan。用于文档检索问题的高效算法。载于第 *13* 届 *SODA* 论文集，第 657–666 页，2002 年。[33] Gonzalo Navarro 和 Sharma V. Thankachan。在有界间隙约束下报告连续子串出现位置。*Theor. Comput. Sci.*，638:108–111，2016 年。[34] Takaaki Nishimoto、Tomohiro I、Shunsuke Inenaga、Hideo Bannai 和 Masayuki Takeda。压缩空间中用于最长公共扩展查询的全动态数据结构。载于第 *41* 届 *MFCS* 论文集，第 58 卷，第 72:1–72:15 页，2016 年。

[35] Wojciech Rytter。将 Lempel-Ziv 分解应用于基于文法的压缩近似。*Theor. Comput. Sci.*，302(1-3):211–222，2003。[36] Peter Weiner。线性模式匹配算法。载于第*14*届 *SWAT* 论文集，第1–11页，1973年。

A 第 2 节中省略的证明

引理 A.1. 给定一个大小为 g、高度为 h 的 ***RLSLP*** G。假设集合 $\mathcal{S}$ 中的每个字符串都是 G 中某个非终结符的展开或其反转的前缀或后缀。$\mathcal{S}$ 的字典树可用 $O(|\mathcal{S}|)$ 的空间实现，以便在 $O(m+\tau\cdot(h+\log m))$ 时间内支持以下查询：给定一个长度为 m 的模式 P，以及 P 的后缀 Q_i，对每个 i，找出按字典序排序的 $\mathcal{S}$ 中以 Q_i 为前缀的字符串区间。

证明。我们先回顾 Karp–Rabin 指纹的定义。

定义 A.2 (Karp–Rabin 指纹**)。** 对于素数 p 和一个 $r \in \mathbb{F}_p^*$，字符串 X 的 Karp–Rabin 指纹 *[24]* 定义为元组 $(r^{|X|-1} \bmod p, r^{-|X|+1} \bmod p, \varphi_{p,r}(X))$，其中 $\varphi_{p,r}(X) = \sum_{k=0}^{|X|-1} S[k]r^k \bmod p$。

我们使用 Christiansen 等人 [8], 的以下结果，该结果建立在 Belazzougui 等人 [3]和 Gagie 等人 [16, 17]的工作之上。

事实 ***A.3*** ([8, 引理 *6.*5]) 。设 $\mathcal{S}$ 为一个字符串集合，并假设我们拥有一个数据结构，能够在 $f_e(l)$ 时间内提取 $\mathcal{S}$ 中字符串的任意长度为 l 的前缀，并能够在 $f_h(l)$ 时间内计算 $\mathcal{S}$ 中字符串的任意长度为 l 的前缀的 ***Karp–Rabin*** 指纹 φ。那么，我们可以构建一个占用 $O(|\mathcal{S}|)$ 空间的数据结构，并支持以下查询，其时间复杂度为 $O(m + f_e(m) + \tau(f_h(m) + \log m))$ ：给定一个长度为 m 的模式 P 及其 $\tau > 0$ 个后缀 Q_1、……、Q_τ（均来自 P），找出按字典序排序的 $\mathcal{S}$ 中以 Q_1、……、Q_τ 为前缀的字符串区间。

需要指出的是，尽管使用了哈希函数，查询算法仍然是确定性的：证明表明，可以在构造期间选择 p 和 r，以确保 $\mathcal{S}$ 中字符串的子串不会发生碰撞。

为了界定 f_e，我们使用 [8, 引理 6。6] 该引理基于 Gąsieniec 等人 [20] 以及 Claude 和 Navarro [9]的工作。

事实 A.4 ([8, 引理6.6])**.** 给定一个大小为 $O(g)$ 的 ***RLSLP***，存在一个大小为 $O(g)$ 的数据结构，使得可以在时间 $f_e(l) = O(l)$ 内，从任意非终结符 A 获取 A 的任意长度为 l 的前缀或后缀。

为了界定 $f_h(l)$，我们引入一个基于以下著名事实的简单构造：

事实 A.5. 设字符串为 X、Y、Z，其中 $XY = Z$。已知这三个字符串中任意两个的 ***Karp–Rabin*** 指纹后，可以在常数时间内计算出第三个字符串的指纹。

断言 A.6. 给定一个大小为 g、高度为 h 的 ***RLSLP*** G，存在一个大小为 $O(g)$ 的数据结构，对于给定的非终结符 A 和整数 l，能够在 $f_h(l) = O(h + \log l)$ 时间内，获取 A^r 和 $\mathrm{rev}(A^r)$ 的长度为 l 的前缀和后缀的 ***Karp-Rabin*** 指纹。

证明。关于 **rev(**A^r**)** 的断言可由关于 A^r 的断言推出：考虑文法 G_{rev}，其中每个产生式中的非终结符顺序均反转。下文重点讨论如何提取 A^r 的指纹，并进一步将讨论范围限制为 A^r 的前缀；后缀的算法类似。

该数据结构由两个集合组成。第一个集合包含文法中所有非终结符展开的长度，第二个集合则包含它们的指纹。

根据事实 A.5 和倍增，只需证明一种计算 A 的长度为 l 的前缀的指纹的算法。假设与规则 $A \to BC$ 相关联的 A。如果 A 的长度小于 l，则返回错误。否则，为了计算 A 的长度为 l 的前缀的指纹，我们分两种情况讨论。如果 $l \le |B|$，则递归处理 B，以获取 B 的长度为 l 的前缀的指纹。否则，递归处理 C，以获取 $C[\ldots l - |B|)$ 的指纹，然后 —

根据事实 A.5，用 B 和 $C[\dots l-|B|)$ 的指纹，可以在常数时间内计算 A 的长度为 l 的前缀的指纹。

对于与规则 $A \to B^r$ 相关联的非终结符 A，我们以类似的方式计算指纹。如果 A 的长度小于 l，则返回错误。否则，设 q 满足 $q \cdot |B| \le l < (q+1) \cdot |B|$。我们通过应用事实 A.5 $O(1+\log q)$ 次，根据 B 的指纹计算 B^q 的指纹，并递归计算 $B[\dots l - q\cdot|B|)$ 的指纹。随后，我们可以应用事实 A.5，在常数时间内计算 A 的长度为 l 的前缀的指纹。注意，在这种情况下，前缀的长度至少缩短为原来的 q。

如果我们遇到终结符 A，计算需要 $O(1)$ 时间（前缀必须等于其自身 A）。

总计，我们花费 $O(h + \log l)$ 时间，因为递归了 $O(h)$ 次；每当我们在某个符号上花费超过常数时间，就将该开销计入长度的缩减中。长度为 l 的后缀的指纹也以类似方式计算。

□

将 $f_e(l)$ 的界（事实 A.5）和 $f_h(l)$ 的界（断言 A.4）代入事实 A.3，即可得到该引理的结论。

□

B 第 3 节中省略的证明

断言 B.1。给定 G' 的一个非终结符 A，我们可以在 A 中找到 P 的最左和最右出现位置，并由此推论，在 $O(\log^2 N \log\log N)$ 时间内找到头部(A) 和尾部(A) 中的对应位置。

证明。我们说明如何在 A 中找到 P 的最左出现位置；最右出现位置可以用类似方法找到。首先，我们通过断言 3.5，在 $O(\log N \log\log N)$ 时间内检查 A 是否包含 P 的出现位置。如果不包含，就可以立即停止。下文假设 A 中存在 P 的出现位置。接着，我们通过断言 3.5，在 $O(\log N \log\log N)$ 时间内检查头部(A)是否包含 P 的出现位置。如果包含，则 A 中 P 的最左出现位置就是头部(A)中 P 的最左出现位置，~~我~~我们可以通过递归处理头部(A)来找到它。如果头部~~(A)不包~~含 P 的出现位置，但 A 包含 P 的相关出现位置，那么 A 中 P 的最左出现位置就是 A 中 P 的最左相关出现位置，我们可以在 $O(|\mathrm{Splits}'(G', P)|) = O(\log N)$ 时间内找到它。最后，如果 P 既不出现在头部(A)中，也没有在 A 中的相关出现位置，那么 A 中 P 的最左出现位置就是尾部(A)中 P 的最左出现位置。如果尾部(A)是一个非终结符 C，我们就递归处理 C 来找到它。如果尾部$(A) = B^{r-1}$ 对于一个非终结符 B，那么尾部(A)不可能包含 P 的出现位置，因为 B 不包含 P，且 A 中不存在相关出现位置。我们最多递归向下处理 $h = O(\log N)$ 层，每层耗时 $O(\log N \log\log N)$。断言得证。

□

引理 B.2. 设 A 是 G' 的一个非终结符。对于任意位置 p，我们可以在 A 中找到 P 的最右出现位置 $q \le p$，并在 A 中找到 P 的最左出现位置 $q' \ge p$，时间复杂度为 $\widetilde{O}(\log^3 N \log\log N)$。

证明。首先，我们描述如何定位 q。考虑解析树中一个标记为 A 的节点 u，该解析树对应于 G'。算法从 u 开始并向下递归。设 A' 为当前节点的标签。它通过断言 B.1 计算 A' 中的最左和最右出现位置、$\mathsf{head}(A')$ 和 $\mathsf{tail}(A')$，以及所有相关出现位置。如果 A' 中 P 的最左出现位置大于 p，则搜索结果为空。否则，考虑两种情况。

1. A' 与规则 $A' \to B'C'$ 相关联，即 $\mathsf{head}(A') = B', \mathsf{tail}(A') = C'$。
   (a) If $p \le |\overline{B'}|$, recurse on B'.

(b) 现在假设 $p > |B'|$。如果 C' 中 P 的最左出现位置小于 p，则对 C' 递归。否则，如果存在，则返回 A' 中 P 的最右相关出现；若不存在，则返回 B' 中 P 的最右出现位置。

2. A' 与规则 $A \to (B')^r$ 相关联，即 $\mathsf{head}(A') = B'$，$\mathsf{tail}(A') = (B')^{r-1}$。设整数 k 满足 $(k-1) \cdot |B'| + 1 \le p \le k \cdot |B'|$。所需的 P 出现位置是以下出现位置中最右边的一个：

   (a) P 的最右出现位置 $q \le p$，它跨越两个 B' 副本之间的边界。为了计算 q，我们先计算 A' 中 P 的所有相关出现位置，然后将每个相关出现位置向右移动可能的最大距离 $r' \cdot |B'|$，其中 r' 是一个整数，可确保其起点位于 p 之前、终点位于 $|A'|$ 之前；再从计算出的出现位置中取最右边的一个，得到 q。

   (b) 满足某个整数 k' 使得 $(k'-1) \cdot |B'| \le q \le q + |P| - 1 \le k' \cdot |B'|$ 成立的 P 的最右出现位置 q（即该出现位置完全属于 B' 的某个副本）。在这种情况下，q 要么是 B' 的第 $(k-1)$ 个副本中 P 的最右出现位置，要么是小于 p 的 B' 的第 k 个副本中 P 的最右出现位置。对于第二种情况，我们通过对 B' 递归来计算 q。

我们最多递归 h 层。每一层中，我们通过断言 B.1 花费 $O(\log^2 N \log\log N)$ 的时间，计算常数个非终结符的最左、最右和相关出现位置及相应偏移。因此，总共花费 $O(h \cdot \log^2 N \log\log N) = O(\log^3 N \log\log N)$ 的时间。

定位 q' 非常相似，只在一些细节技术上有所不同。算法从节点 u 开始并向下递归。设 A' 为当前节点的标签。我们通过断言 B.1，计算 A' 中的最左和最右出现位置、$\mathsf{head}(A')$ 和 $\mathsf{tail}(A')$，以及所有相关出现位置。如果 P 在 A' 中的最右出现位置小于 p，则搜索结果为空。否则，考虑以下两种情况。

1. A' 与规则 $A' \to B'C'$ 相关联，即 $\mathsf{head}(A') = B'$、$\mathsf{tail}(A') = C'$。

   (a) 如果 $p > |B'|$，则对 C' 递归。(b) 现在假设 $p \le |B'|$。如果 P 在 B' 中的最右出现位置大于 p，则对 B' 递归。否则，返回满足条件 $q \ge p$ 的最左相关出现位置 q（如果存在）；若不存在，则返回 P 在 C' 中的最左出现位置。

2. A' 与规则 $A \to (B')^r$ 相关联，即 $\mathsf{head}(A') = B'$，$\mathsf{tail}(A') = (B')^{r-1}$。设整数 k 满足 $(k-1) \cdot |B'| + 1 \le p \le k \cdot |B'|$。所需的 P 出现位置是以下出现位置中最左的一个：

   (a) 横跨 B' 两个副本之间边界的最左出现位置 $q' \ge p$（属于 P）。为计算 q'，我们先计算 A' 中 P 的所有相关出现，然后将每个相关出现向后移动最小可能的距离 $r' \cdot |B'|$，其中 r' 是一个整数，可确保该出现始于 p 之后且止于 $|A'|$ 之前（如果存在），再从计算得到的出现位置中取最左者，得到 q。(b) 满足某个整数 k' 使得 $(k'-1) \cdot |B'| \le q' \le q' + |P| - 1 \le k' \cdot |B'|$ 的 P 的最左出现位置 q'（即该出现位置完全属于 B' 的某个副本）。此时，q' 或者是 B' 的第 $(k+1)$ 个副本中 P 的最左出现位置，或者是 B' 的第 k 个副本中大于 p 的 P 的最左出现位置。对于第二种情况，我们通过对 B' 递归来计算 q'。

时间复杂度与计算 q 的时间复杂度相同。 □

C 第4节中省略的证明

引理 C.1. 假设 P_2 不是 P_1 的子串。经过 $O(m \log N + \log^2 N)$ 时间的预处理后，定理 *3.1* 的数据结构允许在时间 $O(\log^3 N \log\log N)$ 内，计算给定非终结符 A 的展开中 P_1、P_2 的所有 *b-close* 相关共现。

证明。 我们按照定理 3.1 中的说明，在 $O(m \log N + \log^2 N)$ 时间内预处理 P_1, P_2。收到一个非终结符 A后，我们通过断言 B.1，计算 P_1, P_2 在 head(A) 和 tail(A) 中的最左出现位置和最右出现位置，以及 P_1 在 A 中所有相关出现位置的集合 Π_1，和 P_2 在 A 中所有相关出现位置的集合 Π_2。我们将计算 A中的所有相关共现；随后只需从中选出距离为 b 的那些即可。根据 $q_1 \le q_2$ 的定义， A 中 P_1, P_2 的每个相关共现（q_1, q_2）都属于以下某一类别:

1. q_1 是 A 中 P_1 的相关出现位置，且 q_2 是 A 中 P_2 的相关出现位置（即 $q_1 \in \Pi_1$, $q_2 \in \Pi_2$）。要检查一对 $q_1 \in \Pi_1, q_2 \in \Pi_2$ 是否构成 A 中 P_1, P_2 的共现，我们必须检查 q_1 与 q_2 之间是否存在 P_1 或 P_2 的出现位置。 q 只能是 P_2 在 head(A) 中的~~最右出现~~位置、 P_1 在 tail(A) 中的最左出现位置~~,或~~ $\Pi_1 \cup \Pi_2$中的某个出现位置。因此，通过合并两个已排序的集合 $\Pi_1 \cup \{l_q\}$ 和$\{r_q\} \cup \Pi_2$，即可找出此类别中的所有共现；该操作可在 $O(2 + |\Pi_1 \cup \Pi_2|)$ 时间内完成。

2. $1 \le q_1 \le q_1 + |P_1| - 1 \le |$头部($A$)$|$和 $|$头部(A)$| < q_2 \le q_2 + |P_2| - 1$。在这种情况下， q_1必须是 head(A) 中 P_1 的最右出现位置，而 q_2 必须是 tail(A) 中 $q_1 \le q_2$ 的最左出现位置，并且不能存在满足 $q_1 \le q \le q_2$ 的出现位置 $q \in \Pi_1 \cup \Pi_2$。因此，如果此类别中存在共现，我们可以在 $O(|\Pi_1 \cup \Pi_2|)$ 时间内检索到它。

3. q_1 是 A 中 P_1 的一个相关出现（即 $q_1 \in \Pi_1$），并且 $|$头部(A)$| < q_2 \le q_2 + |P_2| - 1$。在这种情况下， q_1 必须是 Π_1 中最右的出现位置，而 q_2 必须是 tail(A) 中 P_2 的最左出现位置，并且 q_1 与 q_2 之间不应存在来自 Π_2 的出现位置。因此，如果此类别中存在共现，我们可以在 $O(|\Pi_1 \cup \Pi_2|)$ 时间内找到它。

4. $q_1 \le q_1 + |P_1| - 1 \le |$头部($A$)$|$和 q_2 是 P_2 在 A 中的相关出现位置（即 $q_2 \in \Pi_2$）。首先，考虑 $q_2 \in \Pi_2$中最左的出现位置。我们通过一次前驱查询，找到 P_1 在 A 中的最右出现位置 $q_1 \le q_2$。当且仅当 ~~P_2~~ ~~在~~头部(A)中的最右出现位置小于 q_1时，(q_1, q_2) 是一个共现，这可以在常数时间内检查。其次，我们考虑 Π_2中其余的出现位置。设 q_2' 为其中最左的一个。我们首先通过一次前驱查询计算前一个 P_1 的出现位置 q_1' ；如果$q_2 \le q_1'$，则输出由此得到的共现。如果 $\Pi_2 = \{q_2, q_2'\}$，则处理完成。否则，根据推论 2.1， $\Pi_2 \setminus \{q_2\}$ 中的出现位置构成一个等差数列，其公差等于 P_2 的周期（因为它们都包含位置 $|$头部(A)$|$）。此外，由于P_1 不包含 P_2，位于 q_2' 之前的 P_1 的出现位置属于由 P_2的相关出现位置构成的周期区域。因此，所有其余的共现都可以通过将周期逐次加到 q_2' 对应的共现上得到。总计，此步骤需要 $O(|\Pi_2| + \log^3 N \log\log N)$ 时间。

□

引理 C.2. 假设 P_2 不是 P_1的子串。可以在 S 中用时 $O(m + (1 +$出现位置$) \cdot \log^4 N \log\log N)$计算 P_1、P_2 的所有 *b-close* 共现位置。

证明。 在预处理过程中，我们对解析树进行剪枝：首先，对于每个非终结符 B，除先序遍历中第一个标记为 B 的节点外，其余节点都转换为叶节点，并剪去其子树。对于

每个标记为非终结符 B 的节点 v，我们存储 anc(v)，即 v 的最近祖先 u，该祖先标记为 A，且 u 是根节点，或 A 在树中标记了多个节点。其次，对于每个标记为非终结符 A 且关联规则 $A \to B^k$ 的节点，我们将其最右的 $k-1$ 个子节点替换为标记为 B^{k-1} 的叶节点。我们将所得的树称为**剪枝语法树**，并为每个标记为非终结符 B 的节点 v 存储 next(v)，即先序遍历中下一个标记为 B 的节点（如果存在）。由于每个非终结符在剪枝语法树中至多标记一个内部节点，且每个节点至多有两个子节点，因此该结构占用 $O(g')$ 空间。

当引理 4.10 的算法输出 $A \in \mathcal{N}'$ 时，我们使用引理 4.11，在时间 $O(\log^3 N \log\log N)$ 内计算 A 中所有相关共现 (q_1, q_2)，并选出满足 $q_2 - q_1 \le b$ 的那些。

固定 A 中一个 b-近距离相关共现 (q_1, q_2)。如果 $\overline{A}$ 与规则 $A \to BC$ 相关联，则构造集合 occ(A) := $\{(q_1, q_2)\}$；否则，如果 A 与规则 $A \to B^k$ 相关联，

$$\mathrm{occ}(A) := \{(q_1 + i \cdot |\overline{B}|, q_2 + i \cdot |\overline{B}|) : 0 \le i \le \lfloor (|\overline{A}| - q_2 - |P_2| + 1)/|\overline{B}| \rfloor\}$$

假设 A 为 G' 的未剪枝语法树中的节点 $v_1, v_2, \ldots, v_k$ 标记。若 W 是一个共现集合，为简便起见，记 $W + \delta = \{(q_1 + \delta, q_2 + \delta) : (q_1, q_2) \in W\}$。下面给出一种生成集合 $\mathcal{S} = \cup_i$occ(A) +off(v_i) 的算法，该集合包含由 (q_1, q_2) 产生的所有次级 b-近距离共现。

我们遍历剪枝语法树，同时维护一个优先队列。队列初始化时，包含先序遍历中第一个标签为 A 的节点，以及 occ(A)。只要优先队列非空，就弹出一个节点 v 和一个集合 W；该集合包含其标签展开中 P_1, P_2 的共现，并执行以下步骤：

- 报告步骤：如果 v 是根节点，则报告 W；
- 下一个节点步骤：如果 next(v) 已定义，则将 (next(v) W + off(next(v)) − off(v)) 推入队列；
- 兄弟节点步骤：如果 v 的标签是非终结符 B，且其兄弟节点的标签是 B^k，其中 k 是某个整数，那么 $W := \cup_{0 \le i \le k} \overline{W} + i \cdot |B|$
- 重复祖先步骤：将 (anc(v) W + off(anc(v)) − off(v)) 推入队列。

根据构造，并且由于每个节点都通过一条 anc 链接路径与根节点相连，该算法会生成 $\mathcal{S}$ 中的所有共现。下面证明它报告 $\mathcal{S}$ 中的每个共现至多一次。假设某个共现 $(q_1', q_2') \in \mathcal{S}$ 被两条不同的步骤序列报告了两次。设 u_1, u_2 为这两条序列中最后一个下一个节点步骤之后被加入队列的节点。我们断言，u_1 是 u_2 的祖先，或反之（否则，u_1 和 u_2 的子树互不相交，因此我们为 u_1, u_2 创建的共现集合的交集为空）。不失一般性，假设 u_1 是 u_2 的祖先。然而，由于我们是通过下一个节点链接到达 u_1 的，根据定义，u_1 的子树必定被剪枝，这就产生了矛盾。

时间复杂度如下：引理 4.10 的算法耗时 $O(m + (1 + \mathrm{occ}) \cdot \log^3 N)$；对 $\mathcal{N}'$ O 中的每个非终结符应用引理 4.11，耗时 (occ $\cdot \log^4 N \log\log N$))；维护队列并报告共同出现位置耗时 O(occ)，因为在每一步中，我们都可以将更新队列所需的时间计入新创建的出现位置，而每个出现位置都会被报告（anc(·) 始终有定义）。 □