arXiv:2601.12200v1 [cs.DS] 18 Jan 2026

计算字符串中的极大重复子序列

Mingyang Gong * AdieshaLiyanage † BraedenSopp ‡
Binhai Zhu §

摘要

本文首次研究如何在单个输入字符串中计算极大（不一定是最长）重复模式。自 2019年 起，Hirota 和 Sakai 仅针对两个或更多输入字符串研究了相关问题（例如，极大公共子序列）。给定一个长度为 n 的输入字符串 S，我们可以在 $O(n\log n)$ 时间内计算 S 的极大平方子序列，大幅改进了计算 S的最长平方子序列所需的 $O(n^2)$ 时间界。对于极大 k-重复子序列，我们的时间界为 $O(f(k)n\log n)$，其中 $f(k)$ 是一个可计算函数，满足 $f(k) < k \cdot 4^k$。对于 $k \geq 3$，这大幅改进了计算S的最长 k-重复子序列所需的 $O(n^{2k-1})$ 时间界。两项结果在受约束情形下仍然成立，即解必须包含 S 的一个子序列 X，但运行时间会更长。

1 引言

计算两个或更多输入字符串中的（非连续）模式是一个经典问题，并已得到广泛应用。教科书中的示例是求两个字符串的最长公共子序列（LCS），其总长度为 n；Wagner 和 Fischer 在 $O(n^2)$ 时间和空间 [14] 内解决了该问题，随后 Hirschberg [6] 又将时间复杂度改进为$O(n^2)$，空间复杂度为 $O(n)$。该问题已在各种应用中得到研究和应用，有时还带有额外约束，例如，LCS 还必须包含模式 X；该模式是同时出现在两个输入字符串中的子序列 A 和 B。如果 A、B 和 X 的长度均为 $O(n)$，Tsai 给出了一种 $O(n^5)$时间算法 [13]。Chin 等人将这一界限改进为 $O(n^3)$ [3]，并且还

*Gianforte 计算学院，蒙大拿州立大学，美国蒙大拿州博兹曼市，邮编 59717。电子邮箱：mingyang.gong@montana.edu。
†Gianforte 计算学院，蒙大拿州立大学，美国蒙大拿州博兹曼市，邮编 59717。电子邮箱：a.liyanaralalage@montana.edu。
‡Gianforte 计算学院，蒙大拿州立大学，美国蒙大拿州博兹曼市，邮编 59717。电子邮箱：braeden.sopp@gmail.com。§Gianforte 计算学院，蒙大拿州立大学，美国蒙大拿州博兹曼市，邮编 59717。电子邮箱：bhz@montana.edu。

由 Chen 和 Chao [2], 进行了研究，后者还额外考虑了这样的版本： A和 B 的 LCS 不得将 X 作为子序列包含在内。

几年前，基于应用中的观察现象——在许多应用中，最长公共子序列有时无法捕捉到真正需要或观察到的模式——Sakai 研究了两个字符串 [10, 11] 的极大公共子序列 ***(MCS)***。对于总长度为 n 的两个字符串 A 和 B，Sakai 可以在 $O(n\log n)$ 时间内计算出 A 和 B 的一个 MCS[1]。后来，对于总长度为 n 的 m 个字符串，Hirota 和 Sakai 提出了一种算法，可在 $O(mn\log n)$ 时间内计算出一个 MCS [4]。这两种算法都适用于解必须包含 A 和 B 的某个模式（或子序列）的情况。

在许多生物学应用中，单个输入字符串中有时会存在有用的非连续模式。例如，植物会经历多达三次全基因组重复；因此，即使随后发生了一些局部突变，仍存在一种普遍的重复模式 [16]。这类应用促使人们研究最长平方子序列 ***(LSS)*** 问题：寻找形如 AA 的最长模式，其中 AA 和 A 本身都是长度为 n 的输入序列 S 的子序列。Kosowski 为此给出了一种 $O(n^2)$ 时间算法 2004 [7]。对于最长的 k-重复子序列，即寻找形如 A^k 的最长模式，其中 A^k 和 A 都是 S 的子序列，有一种朴素算法：只需枚举 S 中的 $k-1$ 个切分位置，然后计算由此得到的 k 个字符串的最长公共子序列。由此得到一种 $O(n^{2k-1})$ 时间算法；遗憾的是，这是目前已知的针对 $k \geq 3$ 的最佳算法。对于 $k = 3$, Wang 和 Zhu 给出了一种参数化的 $O(q^2n^3)$ 时间算法，其中 q 是为了得到一个解 [15] 而从 S 中删除的字母的最小数量。然而，当 $q = O(n)$ 时，该算法的最坏情况运行时间仍为 $O(n^2 \times n^3) = O(n^5)$。

最近，受从串联重复中恢复模式这一问题的启发，Lai 等人研究了最长字母重复子序列（***LLDS***）及相关问题 [9]。这里，字母重复子序列（LDS）是S 的一个子序列，形式为 $\prod_{i=1}^{k} x_i^{d_i}$，其中 x_i 是 S 中的一个字母， d_i 是一个不小于 2 的整数。他们证明，LLDS 可在 $O(n)$ 时间内求解。事实上，这个问题与计算基因组学中的最大游程问题密切相关：在该问题中，每个字母 x_i 至多只能在一个游程（即连续出现）中出现一次，并且 $d_i \geq 1$ [12, 1]。最近，Lafond 等人研究了一个更一般的问题，称为最长子序列重复子序列（***LSDS***）及相关问题，其中，子序列重复子序列（SDS）是 S 的一个子序列，形式为 $\prod_{i=1}^{k} X_i^{d_i}$，其中 X_i 是 S 中的一个子序列， d_i 是一个不小于 2 [8]的整数。他们证明，LSDS 可在 $O(n^6)$ 时间内求解，其基本子程序是计算 S（或其子串）的最长平方序列和立方序列。

在本文中，我们沿着 Hirota 和 Sakai 的研究足迹，开始研究如何计算单个长度为 n 的字符串S 中的各种极大重复子序列。我们的结果总结如下：

[1] [10, 11] 中声称的时间界并不正确，因为没有计入所用数据结构的构建成本 [5]

1. 可以在 $O(n\log\ n)$ 时间内计算出 S 的一个极大平方子序列。2. 对于 $k \geq 3$ 且 $f(k)$ 是关于 k 的可计算函数的情形，即当 $k = O(1)$ 时，可以在 $O(f(k)\ n\log\ n)$ 时间内计算出 S 的一个极大 k-重复子序列；此时运行时间为 $O(n\log\ n)$。

本文组织如下。第 2 节给出必要的定义。第 3 节介绍计算 S 的极大平方子序列的结果。第 4 节介绍计算 S 的极大 k-重复子序列的结果。第 5 节通过提出一些开放问题来总结本文。

2 预备知识

2.1 基础

设 Σ 为一个字母表，S 为字母表 Σ 上的一个给定序列。我们分别用 $n = |S|$ 表示 S 的长度，用 $m = |\Sigma|$ 表示 Σ 的大小。对于每个 $1 \leq i \leq |S|$，我们用 $S[i]$ 表示 S 中第 i 个字符。如果存在 $1 \leq i_1 < i_2 < \ldots < i_{|X|} \leq |S|$ 使得 $X = S[i_1]S[i_2]\ldots S[i_{|X|}]$，则序列 X 是 S 的子序列；在这种情况下，我们也滥用记号，将其写作 $X \subseteq S$。我们用 ϵ 表示空串。给定两个正整数 $1 \leq i < j \leq |S|$，我们用 $S[i, j]$ 表示 S 中从 $S[i]$ 开始到 $S[j]$ 结束的子串。注意，当 $j > i$ 时，我们定义 $S[j,\ i] = \epsilon$。此外，$S[1, j]$ 称为 S 的前缀，而 $S[i, n]$ 称为 S 的后缀。给定一个字符 $\sigma \in \Sigma$，我们用 σ^ℓ 表示由 σ 这个字符串接 ℓ 次得到的序列。我们使用开区间表示法表示索引不包含端点的子串；即，$S(i,\ j) = S[i,\ j-1]$，$S(i,\ j] = S[i+1,\ j]$，以及 $S(i,\ j) = S[i+1,\ j-1]$。此外，我们定义 $\#_S(\sigma)$ 为 S 中 σ 的出现次数。

给定两个序列 S 和 T，S 和 T的极大公共子序列 C 是指这样一个子序列：它不是 S 和 T的任何其他公共子序列 C' 的子序列。（注意，S 和 T 的最长公共序列就是最长的极大公共子序列。）Sakai 算法基于以下观察（该观察在 [10]中并未明确提出）。

观察 2.1 设 C 是两个输入字符串 S 和T的一个公共子序列。C 要么是极大的，要么可以通过插入字母，将 C 扩充为 S 和 T 的一个极大公共子序列 C'。

请注意，这一性质不适用于最长公共子序列；也就是说，无法总是将任意一个公共子序列扩展为最长公共子序列。Sakai 算法 [10]基本上从右向左贪心地搜索S 和 T 的一个公共子序列，并利用额外的极大性性质检查它是否为极大公共子序列；如果不是，就通过添加更多字母进行回溯，直到满足极大性性质（即计算出一个极大公共子序列）。

2.2 单个字符串基础

S 的一个平方子序列 是一个子序列 S ，可表示为 XX的形式。若 S 的平方子序列 B 不被 S 的任何其他平方子序列 C 真包含，则称其为极大的 。作为热身，我们先研究极大平方子序列 问题，该问题旨在求出 S的一个极大平方子序列。可以看出，如果某个字符只出现一次，就可以直接将其从序列中删除，因为它不会出现在任何极大平方子序列中。因此，我们可以假设 S中的每个字符至少出现两次。

定义 2.1 假设 $X = YY$ 是 S 的一个平方子序列。如果存在一个字符 y ，使得 $YyYy$ （或 $yYyY$）是 S 的子序列，我们就称 X 为可右扩展（或可左扩展）。

同理，如果存在一个字符 y ，使得 $Y_1yY_2Y_1yY_2$ 是 S 的子序列，其中 $Y = Y_1Y_2$ ，且 Y_1 和 Y_2 均不等于 ϵ，则称 $X = YY$ 为内可扩展。

显然，当且仅当 X 不可右扩展、不可左扩展且不可内扩展时， X 才是 S 的极大平方子序列。事实上，以下观察对于我们解决最大平方子序列问题的算法至关重要（该算法可视为观察 2.1 的扩展）。

观察 2.2 设 σ 是 S 中出现 ℓ 次的一个字母。那么一定存在一个包含 σ 的 $2\lfloor \ell/2 \rfloor$ 个副本的极大平方子序列。

3 计算极大平方子序列

我们提出一种 $O(n\log\ n)$ 时间算法，用于计算 S 的一个极大平方序列（长度为 n）。我们首先任意选取一个字符 $\sigma \in \Sigma$ ，随后通过扫描序列 S 一次，在 $O(n)$ 时间内记录 S 中每个字符 σ 的位置。 S 中 σ 的位置序列记为 $\langle i_1, i_2, ..., i_\ell \rangle$，其中 S 中 σ 的出现次数为 ℓ 。为便于表述，令$e = \lfloor \frac{\ell}{2} \rfloor$。

我们的算法从平方子序列 $X_0 = \sigma^{2e}$ 开始，首先将其转换为包含 X_0 的平方子序列 X_1 ；此时，序列 X_1 不可右扩展且内可扩展。最后，我们枚举 X_1 的所有可能“最左锚点”（见定义 3.1），并计算一个包含 X_1 且不可左扩展的平方子序列 X_2 。因此，算法生成的最终平方子序列 X_2 是极大的。图1给出了计算 S 的极大平方子序列的算法流程图。

$$X_0 = \sigma^e \cdot \sigma^e \quad \longrightarrow \quad X_1 = YY \quad \longrightarrow \quad X_2 = ZZ$$

图1：算法的流程图。

接下来，我们介绍如何从 X_0 构造 X_1 的步骤。为便于表述，使用记号 $MCS(S_1, S_2, S_3)$ 表示极大公共

子序列，它是 S_1、 S_2 的公共子序列，包含 S_3，由 Sakai 的算法输出。首先，我们计算 $Y = MCS(S[i_1, i_{e+1})\ S[i_{e+1}, n],\ \sigma^e)$。注意， Y 必须是 $S[i_1, i_{e+2}]$ 的子序列，因为 $i_{e+2} > i_{e+1}$，且 Y 是 $S[i_1, i_{e+1}]$的子序列。如果 ℓ 为奇数，且 Y 是 $S[i_{e+2}, n]$的子序列，则通过 $MCS(S[i_1, i_{e+2})\ S[i_{e+2}, n],\ Y)$ 更新 Y。最终生成的 $X_1 = YY$ 是两个 Y 的串接。我们在算法 **1** 中总结了计算 X_1 的过程，并通过一个示例来说明该算法。

算法 1计算 X_1 的算法

输入：一个序列 S、一个重复出现 ℓ 次的字符 σ，以及 $e = \lfloor \ell/2 \rfloor$。输出：一个包含 σ^{2e} 且不可向右扩展、也不可内扩展的平方子序列 X_1。$Y = MCS(S[i_1, i_{e+1})\ S[i_{e+1}, n],\ \sigma^e)$。**如果** ℓ 为奇数且 Y 是 $S[i_{e+2}, n]$ 的子序列**则** $Y = MCS(S[i_1, i_{e+2})\ S[i_{e+2}, n],\ Y)$。返回 $X_1 = YY$。

示例1 我们假设输入序列 $S = abcabcaccabcac$ 和字符 a 被算法选中。字符 a 出现 $\ell = 5$ 次， a 的位置序列为 $\langle 1、4、7、10、13 \rangle$。因此，$e = \lfloor \frac{5}{2} \rfloor = 2$、$i_{e+1} = 7$ 和 $i_{e+2} = 10$。

由于 ℓ 为奇数，因此我们首先计算一个 MCS（$abcabc$、$accabcac$、aa），并且可能的输出 Y可以是 $abcac$。然后运行 MCS（$abcabcacc$、$abcac$、$abcac$），最终输出为 $X_1 = YY = abcac \cdot abcac$。

引理 3.1 给定任意序列 Z， Y 是 Z 的子序列，当且仅当 YY是 ZZ的子序列。

证明。 显然，如果 Y是 Z的子序列，那么 YY是 ZZ的子序列。另一方面，假设 YY 是 ZZ 的子序列，那么第一个 Y 必定是 ZZ 的子序列。我们通过从 ZZ 中贪心地连接 ZZ中的最左字符，构造一个 Y。因此， Y 要么是第一个 Z 的子序列，要么 Y是第一个 Z的超序列。在后一种情况下，第二个 Y是第二个 Z 的某个后缀的子序列——于是 Y必定是 Z的子序列。因此，引理得证。
□ □

引理 3.2 Y 的第一个字符恰好是 σ，即 $Y[1] = \sigma$，其中 $X_1 = YY$。

证明。 我们通过讨论 ℓ 的奇偶性来证明该引理。如果 ℓ 是偶数，那么 Y 包含 σ^e，并且是 $S[i_1, i_{e+1})$ 和 $S[i_{e+1}, n]$ 的极大公共子序列。在这种情况下， $S[i_1, i_{e+1} - 1]$ 中的每个 σ 都必须出现在 Y 中。由于 $S[i_1, i_{e+1})$ 的第一个字符恰好是 σ，因此引理得证。

接下来假设 ℓ 是奇数。如果 Y 是$S[i_1, i_{e+1})$和 $S[i_{e+1}, n]$的极大公共子序列，那么与偶数 ℓ的情况类似，问题便已解决。否则， Y 是 $S[i_1, i_{e+2})$和 $S[i_{e+2}, n]$ 的极大公共子序列；而 Y

包含 σ^e。在这种情况下， $S[i_{e+2}, n]$ 中的每个 σ 都必须出现在 Y中。由于 $S[i_{e+2}, n]$ 的第一个字符恰好是 σ，引理得证。□ □

引理 3.3 包含 X_1 的平方子序列一定不是可右扩展的。

证明。 假设平方子序列 ZZ 包含 $X_1 = YY$。设 ZZ 可右扩展，因此存在一个字符 y，使得 $ZyZy$ 是 S 的平方子序列。根据引理 3.1 和 $\sigma^{2e} \subseteq YY \subseteq ZZ$， σ^e 是 Y 的子序列，且 Y 是 Z 的子序列。因此， $YyYy$ 是 S 的平方子序列，也就是说， YY 也是可右扩展的。接下来考虑以下两种情况。

情况 1： ℓ 是偶数。根据引理 3.2， $Y[1] = \sigma$。由于 Y 包含 σ^e，第二个 Yy 中的字符 $Y[1] = \sigma$ 必须出现在位置i_{e+1}。换言之， $YyYy$ 的第一个 Yy 是 $S[i_1, i_{e+1})$ 的子序列，而第二个 Yy 是 $S[i_{e+1}, n]$ 的子序列。因此， Yy 是 $S[i_1, i_{e+1})$ 和 $S[i_{e+1}, n]$ 的公共子序列，这与 Y 的极大性矛盾。

情况 2： ℓ 为奇数。根据引理 3.2，且 Y 包含 σ^e，第二个 Yy 中的字符 $Y[1] = \sigma$必须出现在位置 i_{e+1} 或 i_{e+2}。换句话说， Yy 必须是 $S[i_1, i_{e+2})$ 和 $S[i_{e+2}, n]$ 的公共子序列；或者是 $S[i_1, i_{e+1})$ 和 $S[i_{e+1}, n]$的公共子序列。

情况 2.1： Yy 是 $S[i_1, i_{e+2})$ 和 $S[i_{e+2}, n]$ 的公共子序列， Y也是。因此， Y 是在算法 1 的第 5 行计算得出的。但由于 Yy的存在， Y并不是 $S[i_1, i_{e+2})$和$S[i_{e+2}, n]$的极大公共子序列，这就产生了矛盾。

情况 2.2： Yy 是 $S[i_1, i_{e+1})$ 和 $S[i_{e+1}, n]$的公共子序列。设 Y'为算法 1 第 3 行生成的序列。如果 Y' 不是 $S[i_{e+2}, n]$ 的子序列，那么 $Y = Y'$。否则， Y 是 Y'的超序列。因此，在两种情况下， Y' 都真包含于 Yy中，这与 Y'的极大性相矛盾。

该引理得证。 □ □

引理 3.4 $X_1 = YY$ 不是内可扩展的。

证明。 反设结论不成立，因此存在一个平方子序列$Y_1yY_2Y_1yY_2$，使得 $Y = Y_1Y_2$，且 Y_1, Y_2 中没有一个是 ϵ。由于 Y_1 不是 ϵ，根据引理 3.2， $Y_1[1] = X_1[1] = \sigma$。与引理 3.3 类似，通过讨论 ℓ的奇偶性即可证明该引理。□ □

根据引理 3.2， $X_1[1] = Y[1] = \sigma$，且 X_1 中的每个副本 Y 都恰好包含 e个σ。因此，当 ℓ 为偶数时， X_1 中的第一个 σ 必须出现在位置 i_1；当 ℓ 为奇数时，则必须出现在位置 i_1, i_2。

定义 3.1（位置 i_t） 假设 Y 是由算法 *1* 生成的序列。给定 σ中的一个位置 i_t， Y 满足 i_t 的最左锚点 是满足以下条件的最小索引 j_t： Y 是 $S[i_t, j_t]$的一个子序列。

注意，索引 j_t 可通过查找以位置 i_t 开头的最左 Y，在 $O(n)$ 中确定，其中位置 i_t 对应于 $t \in \{1, 2\}$。当 $t \geq 3$时，它可能不存在。

示例 2 考虑示例*1*中的输入序列 $S = abcabcaccabcac$ 和字符a 。回顾一下，$Y = abcac$。 Y 在位置 $i_1 = 1$ 处的最左锚点以及 $i_2 = 3$ 分别为 $j_1 = 6$ 和 $j_2 = 8$。

以下算法以算法 1 生成的 $X_1 = YY$为输入，并输出一个极大平方子序列 $X_2 = ZZ$，它是 X_1的超序列。我们首先根据定义 3.1，通过位置 i_1 确定 Y 的最左锚点，其中 i_1 是 S中 σ 的第一个位置。注意， j_1 必定存在，因为 Y 以一个 σ 开头，并且 Y 是 S的子序列。然后我们计算 $Z = MCS(S[1, j_1], S(j_1, n], Y)$。提醒读者，在这种情况下， $S[1, j_1]$以 S 的第一个位置开头；而在算法 1 中，我们总是以 S中 σ 的第一个位置开头。原因是我们希望尽可能多地包含位于 σ 左侧的字符，以确保最终生成的序列不可左扩展。因此，当前的 Z 可能不会以一个 σ开头。注意， ZZ 是一个包含 YY的子序列。根据引理 3.1、3.3和3.4， Z 实际上是某个序列 Z_1 与 Y的串接，即 $Z = Z_1Y$。

接下来说明， Z 在位置 i_2 处的最左锚点必定存在。回顾一下， Z 是 S的子序列（$j_1, n]$， Y也是如此。由于 Y以一个 σ开头，且 $j_1 \geq i_1$，因此 Y必定是 $S[i_2, n]$ 的子序列，所以 j_2 存在。

根据定义 3.1可知， $j_2 \geq j_1$ ，因此 Z 必定是 $S[1, j_2]$的子序列。如果 ℓ 是奇数，且 Z是 S的子序列（$j_2, n]$，那么我们将 Z更新为 $MCS(S[1, j_2], S(j_2, n], Z)$。最终生成的 $X_2 = ZZ$是两个 Z 副本的串接。最后给出算法 2中计算X_2 的伪代码，并在例3中使用示例1和示例 2中的同一实例演示该算法。

算法2 计算极大平方子序列的算法 $X_2$1：输入：由算法 1 生成的序列 $X_1 = YY$。2：输出：包含 X_2 的极大平方子序列 X_1。3：确定 j_1 的最左锚点，其位置为 Y i_1。4： $Z = MCS(S[1, j_1], S(j_1, n], Y)$。5：如果 ℓ 为奇数，则 6：确定 j_2 的最左锚点，其位置为 Y i_2。7：如果 Z 是 $S(j_2, n]$ 的子序列，则 8：$Z = MCS(S[1, j_2], S(j_2, n], Z)$。9：返回 $X_2 = ZZ$。

例3 考虑示例*1*中的输入序列 $S = abcabcaccabcac$ 。算法 *1* 的输出为 $X_1 = YY = abcac \cdot abcac$，其中 $Y = abcac$，字符 a 被算法选取。根据示例 *2*， $i_1 = 1$ 和 $j_1 = 6$。因此，算法 *2* 第*4*行中的序列 Z 由 $MCS[abcabc, accabcac, abcac]$ 输出，并保持为 $abcac$。

注意， $j_2 = 8$ 和 $Z = abcac$ 是 $S(j_2, n] = cabcac$的子序列。因此，最终的 Z由 $MCS[abcabcac, cabcac, abcac]$ 和 $Z = cabcac$计算得出。因此，算法算法*2*为我们提供了一个极大平方子序列

$X_2 = ZZ = cabcac \cdot cabcac$。在此示例中，我们可以看到，第8行对于保证 X_2 是极大的至关重要。

引理3.5 输出的序列 $X_2 = ZZ$ 不可左扩展。

证明。 回顾可知， ZZ 必须是包含 YY的平方子序列。根据引理3.1、3.3和3.4， Z 可以表示为 Z_1Y 的形式，其中 Z_1为某个序列。

我们假设该引理不成立，因此存在一个字符 y，使得 yZ_1YyZ_1Y 是 S 的平方子序列，其中 $Z = Z_1Y$。

情况 1： ℓ 为偶数。注意， Y 是 $S[i_1, j_1]$ 的子序列，而 yZ_1YyZ_1Y的第一个副本 yZ_1Y必须是 $S[1, j_1]$的子序列。因此，$yZ = yZ_1Y$是 $S[1, j_1]$ 和 S的公共子序列（j_1， n]，这与 Z的极大性矛盾。

情况 2： ℓ为奇数。在此情况下，由于 Y 包含 σ^e， yZ_1YyZ_1Y 中第一个副本 yZ_1Y 里的 Y 的第一个 σ 必须出现在位置 i_1 或 i_2。因此，第一个副本 yZ_1Y必须是 $S[1, j_1]$ 和 S的公共子序列（j_1， n] 或；$S[1, j_2]$ 以及 S（j_2， n]。

情况 2.1： yZ_1Y是 $S[1, j_2]$ 和 S的公共子序列（j_2， n]。因此，$Z = Z_1Y$ 是 S的子序列（j_2， n]。由此可知，必须在算法2的第8行计算 Z，这是不可能的，因为 Z 不是 $S[1, j_2]$ 和 S的极大公共子序列（j_2， n]。

情况 2.2： yZ_1Y 是 $S[1, j_1]$ 和 S的公共子序列（j_1， n]。为便于表述，我们令 Z' 为算法2第4行计算出的序列。如果Z' 不是 S的子序列（j_2， n]，那么 $Z = Z'$。否则， Z' 是 Z的子序列。总之， Z' 是 Z的子序列，而 Z被 $yZ = yZ_1Y$ 真包含，这与 Z'的极大性矛盾。

该引理得证。 □ □

引理3.6 输出的序列 $X_2 = ZZ$ 不是内可扩展的。

证明。 假设并非如此，则存在一个字符 y，使得 $Z_1yZ_2Z_1yZ_2$ 是一个平方子序列，其中 $Z = Z_1Z_2$。根据引理3.3和3.4， Z_2 必须包含Y，事实上，等于 Z_3Y，其中某个 Z_3。换言之， $Z_1yZ_3YZ_1yZ_3Y$ 是一个平方子序列，且 $Z = Z_1Z_3Y$。剩余证明与引理3.5的证明类似。 □ □

定理 3.1 给定一个长度为 n 的序列 S，输出的序列 $X_2 = ZZ$是 S 的一个极大平方子序列，可在 $O(n\log\ n)$ 时间内计算得出。

证明。 该定理由 $YY \subseteq ZZ$ 和引理 3.3、3.5 和 3.6 得证。计算的运行时间是因为我们最多调用 Sakai 的 $O(n\log\ n)$ 时间算法 [10] 四次。 □ □

在下一节中，我们将讨论最大 k重复子序列问题。

4 计算字符串中的最大 k-重复子序列

4.1 定义

定义 4.1（对齐）给定两个字符串 A 和 S，A在 S 上的对齐是一个从 $[1,\ |A|]$ 到 $[1,\ |B|]$ 的一对一函数 ϕ，满足 A 中索引的顺序由 ϕ 保持，并且 A 中每个索引处的字母都与 S 中其索引所映射到的字母相同。注意，当且仅当存在 A 在 S上的某种对齐时，A才是 S 的子序列。

定义 4.2（k-重复子序列）设 S 是一个字符串，且令 $k \in \mathbb{Z}_{>0}$。若且唯若 $X^k \subseteq S$，子序列 $X \subseteq S$ 被称为 S 的 k-重复子序列。我们用 k**Rep**（S）表示 S 的所有 k-重复子序列构成的集合。

定义 4.3（最大k-重复子序列）设 S 是一个字符串，且 $X \in k\mathbf{Rep}(S)$。若对于每个满足 $Y \supset X$ 的子序列 Y，都有 $Y \notin k\mathbf{Rep}(S)$，则称 X 是 S 的最大 k-重复子序列。我们用 $\mathbf{M}k\mathbf{Rep}(S)$ 表示 S 的所有最大-k-重复子序列构成的集合。

对于字符串 S、索引 $i \in [1,\ |S|]$ 和 $X \subseteq S$，我们用 $Next_S(X, i)$ 表示满足 $S[i,\ j]$ 包含 X 作为子序列的最小索引 $j \geq i$。如果$X \not\subseteq S[i,\ |S|]$，则定义 $Next_S(X, i)$ 为 Null。类似地，我们用 $Prev_S(X, i)$ 表示满足 $S[l,\ i)$ 包含 X 作为子序列的最大索引 $l \leq i$。如果 $X \not\subseteq S[1,\ i)$，则定义 $Prev_S(X, i)$ 为 Null。直观而言，$Prev_S(X, i)$ 和 $Next_S(X, i)$ 将定义 3.1 扩展到左右两个方向，并使其适用于任意索引。

定义 4.4（σ-起点）给定字符串 S 和 $r \in \mathbb{Z}_{>0}$，σ-起点对于 σ^r 是一个 k 元组 $(p_1,\ p_2,\ \ldots,\ p_k)$，满足以下三个条件：

1. $1 \leq p_1 < p_2 < \ldots < p_k \leq |S|$ 和；2. $S[p_1] = S[p_2] = \ldots = S[p_k] = \sigma$ 和；
3. 为方便表述，令 $p_{k+1} = |S| + 1$。那么，对于每个 $j = 1,\ 2,\ \ldots,\ k$，σ^r 都是 $S[p_j, p_{j+1} - 1]$ 的子序列。

定义 4.5（σ-分割点）给定字符串 S，令 A、$B \in \Sigma^*$，σ-分割点对于 A 和 B 是 $p \in [1,\ |S|]^k$，其中 p 中的每个索引都对应于 S 中的一个 σ，并且对于每个 $i \in [1,\ k]$，$Next_S(B,\ p_i)$ 和 $Prev_S(A,\ p_i)$ 在 Null 中不满足 $Next_S(B,\ p_i) < Prev_S(A,\ p_{i+1})$，当 $i \neq k$ 时。

请注意，σ^r 的 σ-起始位置也是 ϵ 和 σ^{r-1}的 σ-分割点；反过来，任何这样的 σ-分割点都对应于 σ^r的 σ-起始位置。我们将这些定义与以下观察一起，作为算法的一个关键依据。

观察 4.1 令 $X \in \Sigma^*$ 和 $X = A\sigma B$，其中 A、$B \in \Sigma^*$。对于任意满足 $X \subseteq X'$ 的 $X' \in \Sigma^*$，我们有 $X' = A'\sigma B'$，其中某个 $A \subseteq A'$ 和 $B \subseteq B'$。

上述观察告诉我们，如果无法在前缀 A 或后缀 B 中插入字符来扩展 X，那么就不可能在对齐中扩展 X 本身。换句话说， X 的任何扩展都必须通过向左扩展 A 或向右扩展 B 来实现，同时保持中央的 σ 不变。

4.2 概述

本节概述我们的算法。不过，在此之前，我们先考察一个更简单的问题，以帮助理解如何查找 k 重复子序列。

4.2.1 在集合中查找一个极大子序列

与最大平方子序列问题类似，在查找 k 重复子序列时，我们使用子序列关系来比较和排列字符串，而不是依据它们的长度。为了帮助理解我们的算法，我们先考虑这样一个问题：在集合$S = \{A_1, A_2, \ldots, A_k\}$中查找一个没有真超序列的字符串 S（例如，关于子序列关系的极大字符串 S），而不是查找最长字符串。假设我们先从 S 中选取一个字符串 A_i，并将其视为极大字符串 S的候选项。当我们尝试验证 A_i是否为极大字符串时，要么会在 S 中找到某个元素，它是 A_i 的真超序列（因此 A_i 不是极大字符串）；要么找不到这样的元素，因此 A_i 是极大字符串。在第一种情况下，我们可以向 A_i 中插入一些字符，从而得到 S 中一个更大的字符串（就子序列关系而言），并将其作为极大字符串 S的新候选项。因此，重复这一过程，就能找到一些额外的字符，将它们插入 A_i，从而在 S中得到另一个字符串。不过，由于 S 中的字符串数量有限，我们最终必定能在 S 中找到 A_i 的一个超序列，且无法再通过插入其他字符对其进行改进。这意味着，只要在初始猜测的基础上添加字符，总能找到 S 的一个极大子序列。相反，如果我们要查找 S中的最长字符串，这种方法可能行不通，因为 S 中的每个最长字符串都可能要求我们删除 A_i 的部分内容才能得到它。

举个具体的例子，考虑 $S = \{aba, ababb, aa, c, xyxyay\}$。如果从 aa 开始，我们可以在 S 中构造出一个极大字符串：首先通过在 aa 中插入 b，跳转到 aba；然后在 aba 的末尾插入 bb，跳转到 $ababb$。此时可以看出，我们无法再改进 $ababb$，因此找到了一个极大字符串。然而，如果我们要在 S 中寻找最长字符串，那么仅靠插入更多字母，不可能从 aa 得到 $xyxyay$；要做到这一点，就必须从 aa 中删除一个 a。

上面的例子凸显了搜索极大字符串时的另一个特点。如果我们改为选择 c 作为第一个候选项，就会立即在 S 中找到一个极大字符串，尽管 c 也是 S 中最短的字符串。此外，与 aa 不同，在那里我们可能需要经过两轮搜索，寻找要插入的字母，才能得到极大解；而在这里，我们只需要经历

一轮。综合来看，这意味着：(1) 在集合中寻找极大字符串时，我们可以从任意一个字符串开始，寻找可以插入其中的字符，直到它成为一个极大字符串。(2) 初始候选项会影响我们找到解所需的时间。即使第一个候选项很短，选得好也能大幅加快速度。

在介绍所提出的算法之前，我们先描述一种朴素方法，用于寻找字符串 S 中长度为 n 的 k- 重复子序列。假设给定一个候选子序列 $X \subseteq S$，满足 X 是一个 k-重复子序列，即 $X^k \subseteq S$。我们总可以选取在 S 中至少出现 k 次的任意字母 $\sigma \in \Sigma$ 作为初始候选项。注意，任何 k-重复子序列的长度至多为 n/k。对于每个位置，待插入的字母至多有 O（n/k）种选择。

每个扩展后的候选项都可以在 $O(n)$ 时间内验证是否为有效的 k-重复子序列。在每个位置，我们依次尝试每种选择，将字符插入该位置之后，直到无法再插入任何字符，然后继续处理下一个位置。请注意，在较后的位置添加字符，绝不会为已经检查过的位置产生新的候选项。由于候选子序列最多可以扩展 $O(n/k)$ 次，朴素算法的总运行时间为 $O\left(\frac{n^3}{k^2}\right)$。

—

4.2.2 用于 k重复子序列的算法

我们概述该算法如何在给定初始候选项 X in S 的情况下，找到一个新的、用于构成极大 k重复子序列的候选项。概括而言，算法首先选取一个“猜测分隔符”字母 σ in$X$，并将 X 写作 $A\sigma B$，其中 A 是 X 中的前缀，结束于第一个 σ 之前；B 是 X 中的后缀，结束于 S中的第一个 σ 之后。然后，我们尝试通过穷举扫描 S 中 σ 可能出现的所有位置，逐步构建一个包含 A 的新前缀 A'，以及一个包含 B 的后缀B'；在这些位置上，超序列 $A''\sigma B''$ 可能具有相等或更长的后缀和前缀（按照子序列关系判断）。请注意，如果我们验证在任何位置都无法扩展后缀或前缀，那么观察 4.1 表明我们已经得到了极大解。在扫描过程中的每个位置，我们应用 Hirota 和 Sakai 的算法 [4] 来改进当前的 A' 和 B'，并排除在当前位置找到任何k重复子序列 $A''\sigma B''$ 的可能性；若能找到这样的子序列，我们便可通过向前缀或后缀中插入字符来增大 $A'\sigma B'$，并将生成的字符串作为新的候选项。算法穷举完所有位置后，我们便知道，无法在前缀和后缀之前插入任何字符，从而得到一个真超序列，且该超序列在S中同样是 k重复的；因此，当前字符串 $A'\sigma B'$ 是极大的。

现在固定一个字符串 $X \in k\mathbf{Rep}(S)$，它是我们算法的初始候选项。我们可以写作 $X = A\sigma B$，其中 $A \in (\Sigma \setminus \{\sigma\})^*$, $B \in \Sigma^*$，对于任意出现在 X 中的 $\sigma \in \Sigma$ 都成立。令 $r = \#_X(\sigma)$ 表示 σ 在 X 中的出现次数，且 $R = \#_S(\sigma) - k \cdot r$。由于 $X \in k\mathbf{Rep}(S)$，必定存在一种对齐方式，使 X^k 得以对齐，因为它是 S 的子序列。具体而言，X 的每个副本中，σ 的第一次出现都必须映射到 S 中 σ 的某次出现。因此，R 是

σ 在 S 中未用于对齐 X^k。直观地说，我们的算法通过枚举并遍历初始候选项中这些首个 σ 的所有潜在对齐方式来进行。为此，我们考虑将剩余的 σ 分配到 k 个块中的位置：这些块连续放置，下一个块的起始位置由上一个已放置块的位置以及我们选择不放入块中的剩余 $R\,\sigma$ 数量决定。注意，对于首个 σ 的任何潜在对齐方式，每两个连续块中首个 σ 的起始位置之间，至少必须有 σ 个 r；否则，它就不可能是首个 σ 的潜在起始对齐方式。然而，需要注意的是，对于我们考虑的首个 σ 的潜在位置，并不能保证我们可以将 A 对齐到所猜位置上每个 σ 的左侧，并将 B 对齐到其右侧，从而使它们的位置构成 $(A\sigma B)^k$ 的对齐方式。这是因为唯一能够保证的是，在 k 的每对连续块的首个 σ 起始位置之间，S 中至少有 r 次 σ 出现；对于包含 σ^r 的序列（例如 X）中每个块的首个 σ 而言，这是必要条件，却不是充分条件。因此，我们考虑 X^k 中每个块的首个 σ 的所有位置构成的一个超集。给定首个 σ 的一种潜在对齐方式，算法会尝试扩展每个块中相对于首个 σ 的左侧和右侧部分，以形成更大的候选项 k-重复子序列。因此，主要的计算工作在于遍历 X 中首个 σ 的所有潜在对齐方式。幸运的是，我们可以用隔板法论证来界定这类对齐方式的数量。

假设 $Y \in \mathbf{M}k\mathbf{Rep}(S)$，使得 $Y \supseteq X$。那么，在 Y 的每个块中，某个σ（尽管不一定是第一个）的对齐也必然对应于 X中第一个 σ 的某个起始对齐。这意味着，通过系统地考虑 X 中第一个 σ 的所有可能对齐，并尝试使用 Hirota 和 Sakai 的算法 [4], 扩展当前候选项，我们的过程必定能够找到一个包含 X 的最大 k-重复子序列。要计算此类对齐的数量，注意如果 $X \in k\mathbf{Rep}(S)$ 且包含 σ 的 r 次出现，那么对齐 k 个 X 的副本将消耗 S 中 σ 的 $k\cdot r$ 次出现，剩余未使用的 σ 数量为 R。这些剩余的 R 次出现可以按照前文所述，分配到第一个 σ 的位置之间的间隔中，这等价于将 R 个不可区分的项划分到 $k+1$ 个可区分的箱子中。显然，每个块中第一个 σ 的潜在对齐集合与这类划分的集合之间存在一个双射。这为我们提供了一种具体方法，可以界定算法需要进行的猜测次数，并进而推导其最坏情况运行时间。

4.3 算法与正确性

现在，我们将给出一系列引理，用以证明算法正确，并分析其运行时间。

我们首先证明需要生成的 σ-start 数量存在一个上界。在 A 节中，我们提供了一种详细算法，用于快速枚举 σ-starts。

引理 4.1 设 $\ell = \#_S(\sigma)$。不同的 σ-*starts* σ^r 的数量至多为 $\binom{R+k}{k}$，其中 $R = \ell - rk$。

证明。注意 $\ell = kr + R$；因此，$s \in [0,\ k-1]$。令 x_j 表示 $S[p_j,\ p_{j+1}-1])$ 中σ的数量，其中对每个 $j = 1,\ .....,\ k$均成立。根据定义 4.4，我们有$\sum_{j=1}^{k} x_j = \ell$，并且对每个 j都有 $x_j \geq e$。换句话说，如果我们设定 $x'_j = x_j - e$，那么 $\sum_{j=1}^{k} x'_j = \ell - ke = s$ 且 $x'_j \geq 0$。不难看出，不同的 σ-起点数量至多为 $\binom{R+k}{k}$。
□ □

现在我们给出一些与 σ-分割点相关的引理，这些引理可直接由其定义得到。首先，我们将 σ-分割点与 k-重复子序列联系起来。

引理 4.2 设 A、$B \in \Sigma^*$ 和 $\sigma \in \Sigma$。那么，$A\sigma B$ 是 k-重复子序列，当且仅当 A 和 B 存在某个 σ- 分割点。

证明。我们先证明反方向。设 p 是 A 和 B 的某个 σ-分割点。由于对于任意 $i \in [1,\ k]$ $Prev_S$，(A, p_i) 和 $Prev_S(A, p_i)$ 互不为 NULL 和 $S[i] = \sigma$，因此有 $A\sigma B \subseteq S[Prev_S(A, p_i)\ \ Next_S(B, p_i)]$。令 $T_1 = S[Prev_S(A, p_1)\ \ Next_S(A, p_1)]$，且对于 $i \in [2,\ k]$，$T_i = T_{i-1} \cdot S[Prev_S(A, p_i)\ \ Next_S(A, p_i)]$。由于当 $i \neq k$ 时，$Next_S(B, p_i) < Prev_S(A, p_{i+1})$，因此对任意 i 均有$T_i \subseteq S$。于是 $(A\sigma B)^k \subseteq T_k \subseteq S$，并且 $A\sigma B$ 是一个 k-重复子序列。

现在我们考虑正向方向。由于 $A\sigma B$ 是 k-重复子序列的 S，我们有 $(A\sigma B)^k \subseteq S.$，因此存在一个对齐 ϕ，将 $(A\sigma B)^k$ 对齐到 S 上。令 $X = A\sigma B$，则 $p = (\phi(|A\sigma|)\ \phi(|XA\sigma|) \ldots, \phi(|X^{k-1}A\sigma|))$。根据对齐的定义，对于所有 $i \in [2,\ k-1]$，有 $S[\phi(|A\sigma|)] = \sigma$ 以及 $S[\phi(|X^iA\sigma|)] = \sigma$。注意，$\phi(1) \leq Prev_S(A, p_1)$ 和 $Next_S(B, p_k) \leq \phi(|X^k|)$，并且 $\phi(1 + |X^i|) \leq Prev_S(A, p_{i+1})$ 和 $Next_S(B, p_i) \leq \phi(|X^i|)$，对于 $i \in [1,\ k-1]$ 均成立。因此，我们的前驱和后继索引绝不会是 NULL。最后，对于任意 $i \in [1,\ k-1]$，我们有 $Next_S(B, p_i) < Prev_S(A, p_{i+1})$，因为 $Next_S(B, p_i) \leq \phi(|X^i|) < \phi(1+|X^i|) \leq Prev_S(A, p_{i+1})$。因此，$p$ 是 A 和 B 的 σ-分割点。□ □

以下引理阐述了嵌套的 k-相关子序列之间的 σ-分割点。

引理 4.3 设 A、$A' \in \Sigma^*$ 以及 B、$B' \in \Sigma^*$ 满足 $\sigma \in \Sigma$，使得 $A \subseteq A'$ 且 $B \subseteq B'$。A' 与 B' 之间的任意 σ-分割点 p，都是 A 与B之间的 σ-分割点。

证明。注意，当 $A_i = A$ 和 $B_i = B$ 时，这个引理显然成立。设 p 是 A' 和 B' 的一个 σ-分割点。根据定义 4.5，我们知道，对于所有 $i \in [1,\ k]$，$Prev_S(A', p_i)$ 和 $Next_S(B', p_i)$ 均不是 NULL；此外，只要 $i \neq k$，就有 $Next_S(B', p_i) < Prev_S(A', p_{i+1})$。本质上，对于 p中的每个分量 $i \in [1,\ k]$，都存在一些索引 $i_{prev} \leq p_i \leq i_{next}$，使得 $S[i_{prev},\ p_i) \supseteq A'$ 且 $S(p_i,\ i_{next}] \supseteq B'$。由于 $A' \supseteq A$ 且 $B' \supseteq B$，可得 $S[i_{prev},\ p_i) \supseteq A$ 且 $S(p_i,\ i_{next}] \supseteq B$。因此，对于每个$i \in [1,\ k]$，都存在 i'_{prev} 和 i'_{next}，使得 $i_{prev} \leq i'_{prev} \leq p_i \leq i'_{next} \leq i_{next}$

其中 $S[i'_{prev},\ p_i) \supseteq A$ 且 $S(p_i,\ \ i'_{next}] \supseteq B$。由此可直接得出，$Next_S(B,\ p_i) \leq Next_S(B',\ p_i)$ 且 $Prev_S(A',\ p_i) \leq Prev_S(A,\ p_i)$。因此，对于所有 $i \in [1,\ k-1]$，我们有，

$$Next_S(B, p_i) \leq Next_S(B', p_i) < Prev_S(A', p_{i+1}) \leq Prev_S(A, p_{i+1}).$$

由于对于所有 $i \in [1,\ k]$ 和 $Next_S(B,\ p_i) < Prev_S(A,\ p_{i+1})$，索引 i'_{prev} 和 i'_{next} 均存在，因此根据定义 4.5，p 是 A 和 B 的 σ-分割点。□ □

算法 3 扩展重复子序列的算法 要求： X S **的一个** k**-重复子序列，其中** X **不是空串，且字符** σ **出现在** X **中。1：函数** EXTENDkREPSUBSEQ(S, X, σ) **2：** $A \leftarrow X[\ldots i)$ $B \leftarrow X(i, \ldots]$，**其中** i **是 X 中第一个** σ **的索引。3：** $P(\sigma)$ S **的索引，其中** σ **出现。4：对于** $\alpha \in$ ENUMPOTENTIAL-σ-START$(\#_X(\sigma)\ k, P(\sigma))$ **执行 5：如果** α **是** A, B **的** σ**-分割点，则 6：** $e \leftarrow (Prev_S(A, \alpha_2)\ Prev_S(A, \alpha_3) \ldots, Prev_S(A, \alpha_{k-1})\ \ |S|+1)$ **7：** $B \leftarrow$ MkCS$(S(\alpha_1 \ldots e_1) \ldots, S(\alpha_k \ldots e_k);\ B)$ **8：** $s \leftarrow (0,\ Next_S(B, \alpha_1) \ldots, Next_S(B, \alpha_k))$ **9：** $A \leftarrow$ MkCS$(S(s_1 \ldots \alpha_1) \ldots, S(s_k \ldots \alpha_k);\ A)$ **10：返回** $A\sigma B$

现在给出算法 3，用于将 k 中一个非空的重复子序列扩展为 S 中的极大重复子序列，然后展示我们的主要结果。在伪代码中，MkCS 指的是 Hirota 和 Sakai 提出的算法，我们将其记作 MkCS$(S_1, \ldots\ldots, S_k;\ X)$，表示该算法在字符串 S_1 到 S_k 中搜索包含约束 X 的极大公共子序列。函数 ENUMPOTENTIAL-σ-START$(r,\ k,\ I_\sigma)$ 根据按升序给出的 S 中 σ 的位置列表 I_σ，为 S 中的 σ^r 生成所有以 σ 开始的序列。该函数见附录A中的算法4。

定理4.1 设 S 和 $X \in k$**重复**(S) 是字符串，且 σ 出现在 X 中。EXTENDk-REPSUBSEQ$(S,\ X,\ \sigma)$ 在 $X \subseteq Y$ 中返回一个 $Y \in$ Mk **重复**(S)，时间复杂度为 $O\left(\binom{R+k}{k} kn\log\ n\right)$，其中 R 是放置 X 的 k 个副本后，S 中未使用的 σ 的数量。

证明。令 $A_0,\ B_0 \in \Sigma^*$，使得 $X = A_0\sigma B_0$。首先注意，算法 3 执行后，字符串 A 和 B 是 A_0 和 B_0 的超序列，且 $A\sigma B$ 是一个 k-重复子序列。此外，在算法 3 的循环每次迭代后，A 和 B 仍必须是超序列，因为 MkCS 只能生成其约束的超序列。现在注意，算法 3 和算法 3 总是在彼此之间包含一个 σ 的不重叠子串上调用 MkCS，因此 $(A\sigma B)^k$ 包含在这些子串的串接中，故是 S 的子序列。因此，每次迭代后，$A\sigma B$ 都是一个包含 X 的 k-重复子序列

算法 3 因此，该算法输出一个包含 X 的 k-重复子序列。

现在，我们将证明 ExtendkRepSubseq(S, X, σ) 返回的 $A\sigma B$ 是极大的。设 $Y \in \mathbf{kRep}(S)$，使得 $A\sigma B \subseteq Y$。根据观察 4.1，$Y = A'\sigma B'$，其中 $A \subseteq A'$，且对于某些字符串 A'、B'，有 $B \subseteq B'$。进一步根据引理 4.2，我们可知，A' 与 B' 之间必定存在某个 σ-分割点 p。由于 $A_0 \subseteq A \subseteq A'$ 且 $B_0 \subseteq B \subseteq B'$，根据引理 4.3，$p$ 也是 A_0 和 B_0 的 σ-分割点，并且 p 是 $\sigma^{\#_X(\sigma)}$ 的一个 σ-起点。由于 EnumPotential-σ-Start$(\#_X(\sigma), k, P(\sigma))$ 会生成 $\sigma^{\#_X(\sigma)}$ 的所有 σ-起点，因此考虑算法 3 中循环的这次迭代，其中 $\alpha = p$。设这次迭代完成前，算法中 A 和 B 的值分别为 A_p 和 B_p；完成后，它们的值分别为 A'_p 和 B'_p。由于 $B_p \subseteq B$ 且 $A_p \subseteq A$，根据引理 4.3，p 是 A_p 和 B_p 的一个 σ-分割点，因此这次迭代会执行算法 3 到算法 3。由于 $A_p \subseteq A$，对于每个 $i \in [1, k]$，都有 $Prev_S(A', p_i) \leq Prev_S(A_p, p_i)$。算法 3 执行后，由于 $e = (Prev_S(A_p, p_2) \ldots, Prev_S(A_p, p_{k-1}) \quad |S| + 1)$，我们有 $B' \subseteq S(p_i,\ Prev_S(A', p_i)) \subseteq S(p_i,\ e_i)$，其中 $i \in [1,\ k-1]$，且 $B' \subseteq S(p_k,\ |S|+1) = S(\alpha_k,\ e_k)$。因此，$B'$ 是算法 3 中 MkCS 输入的一个公共子序列，其中包含约束 B_p。算法 3 执行后，变量 B 被赋值为 B'_p，它必定是 $i \in [1,\ k]$ 的一个极大公共子序列 $S(\alpha_i,\ e_i)$。由于 $B'_p \subseteq B \subseteq B'$，根据极大性，$B'_p = B'$ 包含 $B = B'$。证明 $A = A'_p = A'$ 的过程非常相似。因此 $A\sigma B = A'\sigma B' = Y$。由此可知，$A\sigma B$ 唯一同时也是 S 的 k-重复子序列的超序列是 $A\sigma B$，所以 $A\sigma B$ 是极大的。

注意，算法 3 在开始迭代循环之前，ExtendkRepSubseq(S, X, σ) 最多耗时 $O(n)$。此外，循环每次迭代所花费的时间主要由扩展 A 和 B所需的时间决定，因此最多耗时 $O(kn\log\ n)$。根据引理 4.1，循环迭代次数不超过 $\binom{\#_S(\sigma) - k\cdot r + k}{k}$。因此，该算法的运行时间为 $O(\binom{R+k}{k} \cdot kn\log\ n)$，其中 $R = \#_S(\sigma) - kr$。□ □

推论4.1 给定一个序列 S，可以在 $O(k\binom{2k-1}{k} \cdot n\log\ n)$ 时间内找到一个极大 k-重复子序列。

证明。如果 S 中没有任何字母重复出现 k 次，我们就能检测到这一点，并在 $O(n + |\Sigma|) = O(n)$ 时间内返回 ϵ。否则，令 σ 为 S中任意一个出现 $\ell \geq k$ 次的字符。接着，我们选取子序列 $X = \sigma^{\lfloor \frac{\ell}{k} \rfloor}$。注意，$X \in k\mathbf{Rep}(S)$显然成立。然后我们调用 ExtendkRepSubseq(S, X, σ)来扩展 X，并在 $O(k \cdot \binom{\ell - k\cdot\lfloor \ell/k \rfloor + k}{k} \cdot n\log\ n)$ 时间内返回一个极大的 k-重复子序列，如 $\ell - k \cdot \lfloor \ell/k \rfloor \leq k - 1$所示。因此，运行时间为 $O(k \cdot \binom{2k-1}{k} \cdot n\log\ n)$。□ □

注意，若我们定义 $f(k) = k \cdot \binom{2k-1}{k}$，则上述时间界变为$O(f(k)n\log\ n)$。此外，当 $k = O(1)$ 时，S的一个极大 k-重复子序列可在 $O(n\log\ n)$ 时间内计算得出。

5 总结

我们给出了高效算法，用于计算长度为 n 的输入字符串 S 的极大平方子序列（或极大 k-重复子序列），运行时间为 $O(n \log n)$（或 $O(f(k)n \log n)$）。这比对应的“最大化”版本快得多；后者的运行时间分别为 $O(n^2)$ 和 $O(n^{2k-1})$。

然而，对于不允许模式 X of S 出现在极大平方子序列（分别为极大 k-重复子序列）中的受约束版本，相关问题仍完全开放。

致谢

本研究部分得到了美国国家科学基金会项目 CNS-2243010 和 CCF-2529539 的支持。

参考文献

[1] Yuichi Asahiro、Hiroshi Eto、Mingyang Gong、Jesper Jansson、Guohui Lin、Eiji Miyano、Hirotaka Ono 和 Shunichi Tanaka。《最长游程子序列问题的近似算法》。载于 Laurent Bulteau 和 Zsuzsanna Lipt´ak（编），第*34*届组合模式匹配年度研讨会，*CPM 2023*，*2023*年*6*月*26*日至*28*日，马恩拉瓦´勒，法国，LIPIcs 第259卷，第2:1–2:12页。Schloss Dagstuhl - Leibniz-Zentrum f¨ur Informatik，2023年。

[2] Yi-Ching Chen 和 Kun-Mao Chao。关于广义受约束最长公共子序列问题。组合优化杂志，21(3):383–392，2011年。[3] Francis Y. L. Chin、Alfredo De Santis、Anna Lisa Ferrara、Ngai Lam Ho 和 S. K. Kim。受约束序列问题的简单算法。《信息处理快报》，90(4):175–179，2004年。[4] Miyuji Hirota 和 Yoshifumi Sakai。求多个字符串的极大公共子序列的快速算法。电子信息通信学会基础电子、通信与计算机科学汇刊，106(9):1191–1194，2023年。[5] Miyuji Hirota 和 Yoshifumi Sakai。枚举两个字符串的极大公共子序列的高效算法。《理论计算机科学》，1048:115360，2025年。[6] Daniel S. Hirschberg。计算极大公共子序列的线性空间算法。*ACM* 通讯，18(6):341–343，1975年。[7] Adrian Kosowski。一种求解最长串联散布子序列问题的高效算法。载于 Alberto Apostolico 和 Massimo Melucci 编，字符串处理与信息检索：第*11*届国际会议，*SPIRE 2004*，帕多瓦，意大利，*2004*年*10*月*5*日至*8*日，会议论文集，《计算机科学讲义》第3246卷，第93–100页。施普林格，2004年。

[8] Manuel Lafond、Wenfeng Lai、Adiesha Liyanage 和 Binhai Zhu。最长子序列重复子序列及相关问题。信息与计算，306:105313，2025年。[9] Wenfeng Lai、Adiesha Liyanage、Binhai Zhu 和 Peng Zou。最长字母重复子序列及相关问题。《信息学学报》，61(3):315–329，2024年。[10] Yoshifumi Sakai。极大公共子序列算法。载于 Gonzalo Navarro、David Sankoff 和 Binhai Zhu 编，组合模式匹配年度研讨会，*CPM 2018*，*2018*年*7*月*2*日至*4*日，中国青岛，LIPIcs 第105卷，第1:1–1:10页。Schloss Dagstuhl - Leibniz 信息学中心，2018年。[11] Yoshifumi Sakai。极大公共子序列算法。《理论计算机科学》，793:132–139，2019年。[12] Sven Schrinner、Manish Goel、Michael Wulfert、Philipp Spohr、Korbinian Schneeberger 和 Gunnar W. Klau。最长游程子序列问题。载于 Carl Kingsford 和 Nadia Pisanti 编，第*20*届生物信息学算法国际研讨会，*WABI 2020*，*2020*年*9*月*7*日至*9*日，意大利比萨（线上会议），LIPIcs 第172卷，第6:1–6:13页。Schloss Dagstuhl - Leibniz 信息学中心，2020年。[13] Yin-Te Tsai。受约束的最长公共子序列问题。《信息处理快报》，88(4):173–176，2003年。[14] Robert A. Wagner 和 Michael J. Fischer。字符串间校正问题。《*ACM*学报》，21(1):168–173，1974年。[15]Lusheng Wang 和 Binhai Zhu。三个字符串的最长公共子序列及相关问题的算法与难度。载于 Franco Maria Nardini、Nadia Pisanti 和 Rossano Venturini 编，字符串处理与信息检索：第*30*届国际研讨会，*SPIRE 2023*，意大利比萨，*2023*年*9*月*26*日至*28*日，会议论文集，《计算机科学讲义》第14240卷，第367–380页。施普林格，2023年。[16] Chunfang Zheng、P. Kerr Wall、James Leebens-Mack、Claude D.E. Pamphilis、Victor A. Albert 和 David Sankoff。全基因组复制后邻域选择作用下的基因丢失及祖先杨属基因组的重建。《生物信息学与计算生物学杂志》，7(03):499–520，2009年。

A 枚举 σ-Start

本节将详细介绍一种生成 σ-starts 的算法。在伪代码中，我们使用**产出** 表示算法会提供一个输出，但不一定会终止；它会暂停并等待请求下一个项，然后继续运行。当没有更多项可提供时，算法便会终止。首先注意，
EnumPossibleDivisions 会产出 $\Delta_{d;h} = \{\bar{x} \in \mathbb{Z}_{\geq 0}^{d+1}$ 的每个元素：
$\sum_{i=1}^{d+1} x_i = h\}$ 各一次，并且每产出一个项所需的时间不超过$O(d)$。直观而言，这个函数等价于生成将 h 个不可区分的元素划分到 $d+1$ 个可区分的箱子中的所有方式。

算法4用于枚举在 d个 k 对象之间进行划分的算法
分隔物
要求： k 和 d 是正整数。

```
函数 EnumPossibleDivisions(k, d)
    如果 d ≤ 0 则
        返回 k
    对于 i ← 0 到 k，执行
        对于 sol ∈EnumPossibleDivisions(k − i, d − 1) 执行
            返回 [i] + sol
```

要求： k 是 S 中 X 的重复次数，r 是 σ 的数量。I_σ是 S 中每个 σ 的索引列表，按升序排列。**函数** EnumPotential-σ-Starts(r, k, I_σ) **对于** $sol \in$ EnumPossibleDivisions$(|I_\sigma| - k \cdot r,\ k)$ **执行** **返回** $(I_\sigma[(t-1) + 1 + \sum_{i=1}^{t} sol[i]]$ 对于 t 从 1 到 $k-1)$

引理 A.1 算法 EnumeratePotential-σ-Starts 会为 $\sigma^r \subseteq S$ 中的每个 σ-*start* 各生成一次，并且生成每个值总共耗时 $O\left(k \cdot \binom{\#_S(\sigma) - k \cdot r + k}{k}\right)$。

证明。令 $R = \#_S(\sigma) - k \cdot r$，它等于将 k 个 σ^r 副本放入 S 后剩余的未使用 σ 的数量。考虑 $\boldsymbol{\delta} \in \Delta_{k,\ R}$ 并定义 $w \in \mathbb{Z}_{\geq 0}^{k}$ 如下：

$$w_1 = \delta_1, \quad \text{and} \quad w_{i+1} = w_i + \delta_{i+1} + r \quad \text{for } i \in (1,\ k].$$

注意：

$$w_k = \sum_{i=1}^{k} \delta_i + r(k-1) + 1.$$

因此：

$$\#_X(\sigma) - w_k \geq r - 1.$$

这表明，在 S中 σ 的第 w_k 次出现之后，至少还剩下 $r-1$ 次出现。

对于每个 $i \in [1,\ k)$，我们可以选取 S 中 σ 的第 w_i 次出现，并构建 σ^r的对齐。由于 $w_{i+1} - w_i \geq r$，从第 w_i 次出现处开始的 σ^r 块将在第 w_{i+1} 次出现之前结束。

此外，注意：

$$w_i = \sum_{j=1}^{i} \delta_j + r(i-1) + 1,$$

这正是算法4中的算法4所出现的表达式。因此，过程
EnumPotential-σ-Starts 总能为 σ^r生成一个有效的 σ-起始。

此外，对于 σ^rk的每个对齐 ψ，我们都可以构造出（k+1）元组 $q \in \mathbb{Z}_{\geq 0}^{k+1}$ $q_1 = |\{i \in H_\sigma \ \ i < \psi\ 1\ \}|$ $q_{k+1} = |\{i \in H_\sigma \ \ \psi\ 1 + r\ k - 1 \ \ <$，其中：$i\}|$ $q_i = |\{j \in H_\sigma \ \ \psi\ 1 + r\ i - 2 \ \ < j < \psi\ 1 + r\ i - 1 \ \ |\}$ H_σ ( ) , : (( )) ，以及：(( )) (( ))，其中是 σ 在 S 中未被对齐 ψ 使用的索引集合。显然，$q \in \Delta_{k,\ R}$，而且可以根据 q唯一确定 ψ 中每个 σ^r 块内第一个 σ 的位置。因此，每个 σ-起始恰好生成一次。

请注意， EnumPotential-σ-Starts对$\Delta_{k,R}$中的每个元素恰好生成一个唯一输出，每个元素的处理时间为 $O(k)$。由于$|\Delta_{k,R}| = \binom{R+k}{k}$，
EnumPotential-σ-Starts 生成每个起始点需要 $O\left(k \cdot \binom{R+k}{k}\right)$ 时间。□

□

根据引理 4.3 和前面的引理，我们得到以下结论：

推论 A.1 算法 EnumeratePotential-σ-Starts为重复子序列 $X \subseteq S$求出 $\sigma^{\#x}$ (σ) 的每个σ-起始点。