

# 全动态模型下最短唯一回文子串问题的数据结构与算法

申治洺

2024 年 5 月 20 日

## 摘要

本文在 Data Structures for Computing Unique Palindromes in Static and Non-Static Strings[1]（以下称原文）的基础上，继续探讨了最短唯一回文子串问题的求解。本文使用较大篇幅概述原文，尤其是 Inoue 算法 [2] 及其改进，这是求解最短唯一回文子串问题的基础。本文调研了回文子串领域的研究成果，主要包括最短唯一回文子串、极小唯一回文子串、极大回文子串这三个方面。最后，本文针对原文仅支持两种半动态模型，不支持全动态模型，适用范围过于狭窄的缺陷，提出了一种全动态模型，并提出了支持在动态串的任意位置插入和删除操作的求解最短唯一回文子串问题的算法。该算法的空间复杂度为  $O(n)$ ，初始构建数据结构的时间复杂度为  $O(n)$ ，应对插入或删除操作，更新数据结构并计算最短唯一回文子串的时间复杂度为  $\tilde{O}(m)$ ，其中  $n$  为原始串的长度， $m$  为被操作位到串尾的距离。

## 关键词

- 最短唯一回文子串 (shortest unique palindromic substring, SUPS)
- 极小唯一回文子串 (minimal unique palindromic substring, MUPS)
- 极大回文子串 (maximal palindrome)
- (全) 动态模型 (dynamic model)

## 1 原文概述

本文在概述原文时，着重介绍数据结构的设计。对于原文中涉及的大量引理 (Lemma)、定理 (Theorem)、命题 (Proposition)、推论 (Corollary)，本文直接引用其中的结论，不再关注其证明过程，因为原文提供了详尽的证明或证明所在的参考文献。由于原文中直接引用了一些不太常用的数据结构，

如后缀树、van Emde Boas 树等，本文会适当补充相关数据结构的定义、实现、性质。对于原文中定义不够明确的概念，本文给出了严格的定义，并描述和论证了相关性质。

## 1.1 问题引入

**MUPS** 给定串  $T$ ,  $1 \leq i \leq j \leq |T|$ , 回文子串  $u = T[i..j]$ , 如果  $u$  在  $T$  中是唯一的,  $u' = T[i+1..j-1]$  在  $T$  中不唯一, 则称  $u$  是  $T$  的极小唯一回文子串 (minimal unique palindromic substring).

**interval SUPS** 给定串  $T$  和区间  $[p, q]$ , 回文子串  $v = T[i..j]$ ,  $v$  在  $T$  中唯一且包含区间  $[p, q]$ , 任何更短的包含区间  $[p, q]$  的  $T$  的回文子串都在  $T$  中不唯一, 则称  $v$  是对区间  $[p, q]$  的  $T$  的最短唯一回文子串 (shortest unique palindromic substring), 记作  $v = \text{SUPS}_T([p, q])$ .

**point SUPS** 区间 SUPS 问题在  $p = q$  的情况下退化为点的 SUPS 问题, 记作  $v = \text{SUPS}_T(p)$ .

**ME** MUPS 向外扩展, 即每次起点向前移动一位, 终点向后移动一位, 直至破坏回文性为止。在此过程中可以得到“极小唯一回文子串的极大扩展” (maximal extension of MUPS).

**MP** 对任意一个半整数, 即  $1, 1.5, 2, 2.5 \dots, n-1$ , 都能求得以其为中心的极大回文子串 (maximal palindrome).

需要注意的是, MUPS 是极小的, 强调的是 MUPS 都无法在自身基础上再缩小一点, 因为再缩小会违背 MUPS 的唯一性, MUPS 的定义并不关心每个 MUPS 与其他 MUPS 的长度比较。SUPS 是最短的, SUPS 的定义基于多个包含给定区间或点的回文子串的长度比较, 但这并不意味着 SUPS 仅有 1 个。原文中的 Theorem 1 说明 SUPS 至多只有 4 个。

由 MUPS 的定义可知, MUPS 具有唯一性, 此外 MUPS 不存在嵌套的情况, 可用反证法论证如下:

取 MUPS  $M_1 = T[s_1..e_1]$ , MUPS  $M_2 = T[s_2..e_2]$ , 满足  $s_1 \leq s_2 \leq e_1 \leq e_2$ , 且  $s_1 \leq s_2$  和  $e_1 \leq e_2$  不能同时取等。找到  $M_1$  的对称轴  $l$ , 若  $M_2$  关于  $l$  对称, 则  $M_1$  显然不满足 MUPS 定义中的极小性; 若  $M_2$  不关于  $l$  对称, 则

<sup>1</sup>原文中串从 1 开始标号, 本文也采用这种规范。

取串  $M_3$  关于轴  $l$  与  $M_2$  对称, 由于  $M_2$  是回文串,  $M_3$  与  $M_2$  完全一致, 显然违背了  $M_2$  作为 MUPS 的唯一性。

原文中使用的“极大回文子串”(maximal palindrome)一词, 其实兼指 ME 和 MP。这里明确地区分了这两个概念, 并在下文探讨两者的性质和联系。

ME 具有唯一性和非嵌套性, 可用反证法论证如下:

假设存在两个完全一致的 ME  $m_1$  和  $m_2$ ,  $m_1$  对应的 MUPS 为  $M_1$ , 以  $m_1$  和  $M_1$  的相对位置关系在  $m_2$  中找到回文子串  $M_2$ ,  $M_2$  一定与  $M_1$  不重合, 于是违背了  $M_1$  作为 SUPS 的唯一性。ME 的唯一性得证。

取 ME  $m_1 = T[s_1..e_1]$ , ME  $m_2 = [s_2..e_2]$ , 满足  $s_1 \leq s_2 \leq e_1 \leq e_2$ , 且  $s_1 \leq s_2$  和  $e_1 \leq e_2$  不能同时取等。找到  $m_1$  的对称轴  $l$ , 若  $m_2$  关于  $l$  对称, 则  $m_2$  不满足 ME 定义中的极大性; 若  $m_2$  不关于  $l$  对称, 则取串  $m_3$  关于轴  $l$  与  $m_2$  对称, 由于  $m_2$  是回文串,  $m_3$  与  $m_2$  完全一致,  $m_2$  不满足上文已证的 ME 的唯一性。ME 的非嵌套性得证。

显然, MP 完全包含了 ME。通常情况下, MP 包含了大量的空串和单字符串, 并有大量重复情况。例如, 串  $T = \text{babbbabbababb}$ , 以 2 为中心的 MP bab 和以 11 为中心的 MP bab 发生重复, 因此两者都不包含在 ME 中。

事实上, 唯一且非空的 MP 等同于 ME。证明如下:

对任意非空的 MP, 定义“缩小串集”为 MP 缩小<sup>2</sup>过程中产生的所有回文串的总和。易知, “缩小串集”包含了所有回文子串, 也就包含了所有 ME。不唯一的 MP 缩小产生的回文串也是不唯一的, 因而不能产生 ME。对唯一的 MP 进行缩小, 缩小的尽头是空串, 必然丧失唯一性。因此, 在缩小的过程中一定存在唯一性发生改变的临界, 即可找到一个 MUPS, 则该唯一的 MP 也就成为一个 ME。综上, 唯一且非空的 MP 与 ME 一一对应。

### 1.1.1 最长公共扩展 (LCE) 与后缀树 (suffix tree)

对于长度为  $n$  的串  $S$ , 后缀树 (suffix tree) 定义为这样一棵树, 满足如下五个条件<sup>3</sup>:

- 有  $n$  个子节点, 分别编号 1 至  $n$
- 除根节点外, 每个非叶节点都有至少两个子节点

<sup>2</sup>类似于扩展, 缩小指每次将回文串的起点向后移动一位, 终点向前移动一位。

<sup>3</sup>后缀树的定义和图 1 引自 Suffix tree Wikipedia, 可参见 [https://en.wikipedia.org/wiki/Suffix\\_tree](https://en.wikipedia.org/wiki/Suffix_tree)

- 每条边都用  $S$  的一个非空子串来标记
- 起始于同一节点的两条边不能有同一字符开头的标记
- 从根节点到叶节点  $i$ , 将所有边的标记依次拼接起来, 可得到后缀  $S[i..n]$

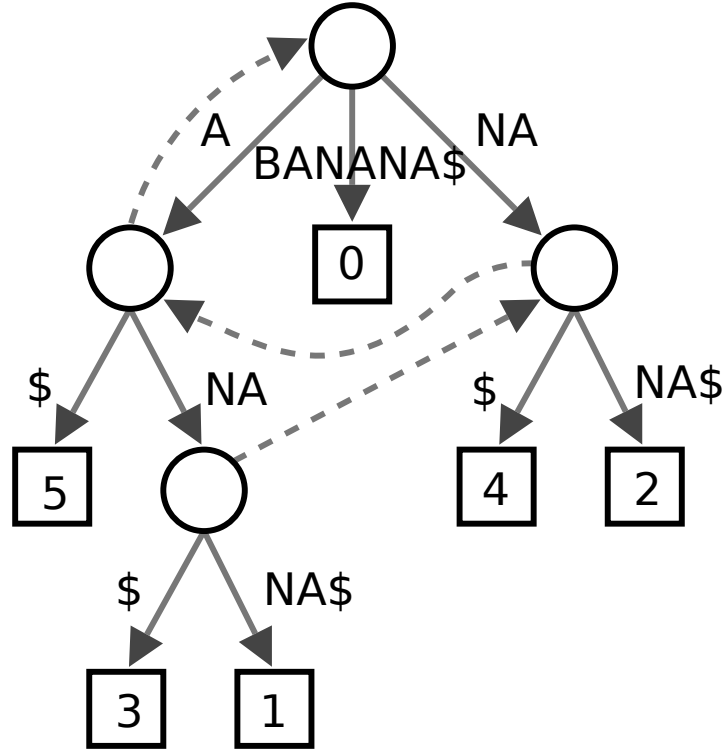


图 1: BANANAS\$ 的后缀树, 虚线为构建过程中的辅助线

最长公共扩展 (longest common extension, LCE) 问题是对一个给定的串  $T$ , 给定的整数  $i, j$ , 满足  $1 \leq i, j \leq n$ , 计算出两个后缀  $T[i..n]$  和  $T[j..n]$  的最长公共前缀的长度。LCE 问题可以通过构建串  $T\$$ <sup>4</sup> 的后缀树<sup>5</sup>在常数时间内求得 [3]。我们可以构造一个双向 LCE 数据结构, 即对串  $T\#T^R\$$  构造后缀树, 从而实现在常数时间内求得:

- $T$  的任意两后缀的最长公共前缀

<sup>4</sup>这里的  $\$$ , 图 1 中的  $\$$ , 以及下文出现的  $\#$ , 都指不属于字母表  $\Sigma$  的特殊字符。

<sup>5</sup>解决某一问题的常用数据结构可用问题名代指。例如, 这里的数据结构可称为 LCE 数据结构。

- $T$  的任意两前缀的逆的最长公共前缀
- $T$  的任意后缀和  $T$  的任意前缀的逆的最长公共前缀

由第三点易知, 对任意位  $c$ , 将  $T$  分割为前缀  $T_1$  和后缀  $T_2$ , 我们可以用双向 LCE 数据结构求得  $T_2$  和  $T_1^R$  的最长公共前缀, 即以  $c$  为中心的极大回文子串。

### 1.1.2 区域最小值请求 (RmQ) 及其变体 LogRmQ

区域最小值请求 (range minimum query, RmQ) 是对一个给定的整型数组  $A$ ,  $A$  上的两个下标  $i, j$ , 满足  $i \leq j$ , 计算出下标  $k$  使得  $A[k]$  为  $A[i..j]$  上的最小值。需要注意的是, 有时最小值有多个, 而 RmQ 每次只能找到其中一个。存在这样一个数据结构, 可以在  $O(n)$  时间内构建, 并在  $O(1)$  时间内回应任一个 RmQ。

对于一个 RmQ, 如果任何请求的范围宽度都满足  $O(\text{polylog}(n))$ , 则称这样的 RmQ 为 LogRmQ。

## 1.2 前驱 (predecessor)、后继 (successor) 与 van Emde Boas 树

对一个不减的整型数组  $B$ , 给定整型数  $x$ , 定义前驱 (predecessor) 是小于  $x$  的最大值; 同理, 定义后继 (successor) 是大于  $x$  的最小值。我们使用 van Emde Boas 树求解前驱和后继。van Emde Boas 树的空间复杂度为  $O(U)$ , 各项操作的时间复杂度为  $O(\log \log U)$ , 其中  $U$  为全局大小 (universal size)[4]。在回文串领域, 取  $U = n$ 。

### 1.3 Inoue 算法概述

Inoue 算法 [2] 的数据结构由下面五部分组成:

- 所有 MUPS 的集合
- 所有 MP 的集合
- 对所有 MUPS 开始位置组成的数组求后继的数据结构
- 对所有 MUPS 结束位置组成的数组求前驱的数据结构

- 所有 MUPS 的长度,按照 MUPS 的开始位置排序,组成的数组 MUPSlen, 以及对该数组的 RmQ 数据结构

对给定的区间  $[p, q]$ , 通过分别对相应数组求  $p$  的前驱和  $q$  的后继, 考察区间  $[p, q]$  覆盖 MUPS 的情况:

1.  $[p, q]$  只覆盖 1 个 MUPS, 则该 MUPS 的能覆盖区间  $[p, q]$  的最小扩展即为所求。
2.  $[p, q]$  覆盖多个 MUPS, 则无解, 因为任何 SUPS 最多只能包含一个 MUPS[2]。
3. 找不到这样的 MUPS, 再分为如下三种情况。
  1. 基于  $q$  前结束的最右边的 MUPS  $M_l$  的扩展
  2. 一个覆盖  $[p, q]$  的 MUPS
  3. 基于  $p$  前开始的最左边的 MUPS  $M_r$  的扩展

依次分析求解即可。其中要利用所有 MP 的集合求解特定中心的 MP, 以及递归地使用 RmQ 以求出所有 SUPS。所有 MP 的集合和所有 MUPS 的集合可在  $O(n)$  时间内求得 [2, 5], 同时可构造完相应的前驱、后继、RmQ 数据结构, 且该算法数据结构五部分的空间复杂度都为  $O(n)$ , 所以总的来说, 空间复杂度为  $O(n)$ , 预处理的时间复杂度为  $O(n)$ . 设共有满足条件的 SUPS  $\alpha$  个, 该算法中各个元操作的时间复杂度都为  $O(1)$ , 且  $\alpha$  决定了最后 RmQ 递归搜索的次数, 所以该算法求解 SUPS 问题的时间复杂度为  $O(\alpha)$ .

#### 1.4 原文对 Inoue 算法的改进

原文对 Inoue 算法的改进可总结为如下几个方面:

- 不再保留所有 MP 的集合, 而是使用 1.1.1 中的双向 LCE 数据结构求解特定中心的 MP。但仍需  $O(n)$  的空间和  $O(n)$  时间预处理。
- 使用位数组<sup>6</sup> $B, E, L$  分别保存 MUPS 的起点、MUPS 的终点、ME 的起点。ME 的终点不需要保存, 因为 ME 和 MUPS 一一对应, 可由 MUPS 的终点和 MUPS 起点与 ME 起点之差计算出 ME 的终点。但仍需  $O(n)$  的空间和  $O(n)$  时间预处理。

<sup>6</sup>只有 0 和 1 两种元素的数组, 类似于 Java 中的 boolean 型数组。

- 证明不等式  $|\text{SUPS}_T([p, q])| \leq 4$  恒成立 (原文中的 Theorem 1), 即  $\alpha \leq 4$ , 故  $O(\alpha) = O(1)$ .

### 1.5 原文算法的示例

可参见原文 Fig.6。

## 2 领域综述

回文串是从前向后和从后向前读取结果相同的序列。由于其独特的性质以及在模式识别、数据压缩和 DNA 序列分析中的广泛应用, 在生物信息学、计算机科学等领域, 对回文串的研究不断。本节探讨了回文串的主要研究方向, 包括但不限于极小唯一回文子串、最短唯一回文子串和极大回文子串。

### 2.1 极小唯一回文子串

极小唯一回文子串 (MUPS) 的定义和性质详见 1.1。求解 MUPS 的算法设计策略总结如下:

- 基于后缀树。对输入的串构建后缀树, 从后缀树中获取所有极小唯一的子串, 再从中提取回文串 [3]。这种策略需要消耗  $O(n)$  的时间以完成构建后缀树的预处理 [6]。
- 基于动态规划。使用一个动态规划的表格标记出所有回文子串, 再依次检查每个回文子串是否唯一。时间复杂度为  $O(n)$ [5]。

### 2.2 极大回文子串

极大回文子串 (MP) 的定义和性质详见 1.1。求解 MP 的算法设计策略总结如下:

- 使用 Manacher 算法 [5]。这是一个求解 MP 的在线 (on-line) 算法<sup>7</sup>, 可以在线性时间内获得结果。
- 使用 Eertree (Palindromic Tree)[7]。这种数据结构可以储存所有互异回文子串, 以便快速求解 MP。

---

<sup>7</sup>即支持一位一位地按顺序输入串, 运行过程中的结果对于已输入串的问题解。

## 2.3 最短唯一回文子串

最短唯一回文子串 (SUPS) 的定义和性质详见 1.1。求解 SUPS 问题, 可以使用 Inoue 算法 [2], 详见 1.3; 以及原文 [1] 对 Inoue 算法的改进算法, 详见 1.4 和 1.5。

## 2.4 回文分解

回文分解 (palindromic factorization) 是将一个串划分为最少数量的回文串的过程, 常用于数据压缩和文本挖掘。借助 Eertree 数据结构,  $k$ -分解问题可以在  $O(kn)$  时间内完成 [7]。

# 3 当前算法的不足

原文基于 Inoue 算法提出了区间 SUPS 问题和点的 SUPS 问题的优化算法, 只需要  $O(n)$  空间的数据结构,  $O(n)$  的预处理时间, 即可在常数时间内计算出 SUPS 问题的解。如果只使用大  $O$  记法, 不考虑时空复杂度的系数, 那么这两类问题的算法已经难以继续优化了。

原文的后半部分提出了两种半动态模型 (semi-dynamic model), 并据此初步探讨了动态情形下 SUPS 问题的求解。

**滑窗模型** 每次支持的操作为, 删除最左端的一位, 或在最右端添加一位。滑窗模型构建了一个不断向右移动的窗, 期间窗的大小<sup>8</sup>可能发生变化。

**替换模型** 每次支持替换串中的一位为字母表<sup>9</sup>中的另一个字符。

原文中为滑窗模型提出了一个相当高效的算法, 支持在每次操作后以均摊的  $O(\log \sigma + \log \log W)$  时间调整数据结构, 并在  $O(\log \log W)$  求解任何 SUPS 问题。原文中为替换模型提出的算法也相当高效, 可以在  $O(\log n \log \log n + k \log \log n)$  时间<sup>10</sup>内解决任意  $k$  个 SUPS 问题。这两种算法的空间复杂度和预处理过程的时间复杂度与静态模型一致。

在半动态模型下, 被操作位的位置和操作方式都受到较大限制, 因而很难满足实际应用的需求。本文基于原文中的两个半动态模型提出全动态模型, 支持对原串进行如下三种操作:

<sup>8</sup>原文中窗的大小用  $W$  表示, 本文也采取这种规范。

<sup>9</sup>原文中字母表的大小用  $\sigma$  表示, 本文也采取这种规范。

<sup>10</sup>已包含调整数据结构的时间。



- 对原串中任一个间隔，以及开始处和终止处，插入一个新位
- 将原串中任一位置删除
- 将原串中任一位置替换，即用字母表中的其他字母来取代它

## 4 改进算法的提出

本节为 3 提出的全动态模型设计实现算法。原文 6.2 节已实现支持任一位置的替换的半动态模型，所以这里只需实现任一位置的删除和添加操作的情形。

### 4.1 删除情形

由于计算 MUPS 集合 [5] 和 LCE[6] 的算法都是在线的。可以对内部细微数据结构（如后缀树中的节点和边）进行标记，从而可以在后面的任意时刻退回到刚读入第  $i$  位时的状态  $S_i$ 。设外部请求删除第  $k$  位，则立刻让 MUPS 集合和 LCE 退回状态  $S_{k-1}$ 。又由 [8] 知每次在串尾插入一位，改变的 MUPS 的数量为常数，且可以在均摊的  $O(\log \sigma)$  时间内检测到变化并调整 MUPS 集合。从原来的第  $k+1$  位开始重新插入，共经  $m = n - k$  次插入，若忽略对数因子，则可以在  $\tilde{O}(m)$  时间内完成对数据结构的调整。

### 4.2 添加情形

仿照 4.1，设外部请求在第  $k$  位和第  $k+1$  位的间隔处添加一位，则立刻让 MUPS 集合和 LCE 退回状态  $S_k$ 。插入新位，再从原来的第  $k+1$  位开始重新插入，共经  $m = n - k$  次插入，若忽略对数因子，则可以在  $\tilde{O}(m)$  时间内完成对数据结构的调整。

### 4.3 回顾与展望

本节实现的  $\tilde{O}(m)$  时间复杂度的算法适用于被操作为靠近串尾的情况，如果不满足该条件，则实际的时间复杂度与  $O(n)$  并无太大差别。而实际上，每次重新计算 MUPS 集合、MP 集合或 LCE 也只需要  $O(n)$  的时间。原文中能设计出比  $O(n)$  高效的算法，得益于论文 [8, 9] 提出了特定情形下修改

MUPS 集合的高效算法，而这两篇论文中的算法都基于 Eertree[7] 数据结构。未来实现高效的全动态模型 SUPS 算法可能还是要从 Eertree 着手。

## 参考文献

- [1] T. Mieno and M. Funakoshi, “Data structures for computing unique palindromes in static and non-static strings,” *Algorithmica*, vol. 86, no. 3, pp. 852–873, 2024. [Online]. Available: <https://doi.org/10.1007/s00453-023-01170-8>
- [2] H. Inoue, Y. Nakashima, T. Mieno, S. Inenaga, H. Bannai, and M. Takeda, “Algorithms and combinatorial properties on shortest unique palindromic substrings,” *J. Discrete Algorithms*, vol. 52-53, pp. 122–132, 2018. [Online]. Available: <https://doi.org/10.1016/j.jda.2018.11.009>
- [3] D. Gusfield, *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997.
- [4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009. [Online]. Available: <http://mitpress.mit.edu/books/introduction-algorithms>
- [5] G. K. Manacher, “A new linear-time ”on-line” algorithm for finding the smallest initial palindrome of a string,” *J. ACM*, vol. 22, no. 3, pp. 346–351, 1975. [Online]. Available: <https://doi.org/10.1145/321892.321896>
- [6] E. Ukkonen, “On-line construction of suffix trees,” *Algorithmica*, vol. 14, no. 3, pp. 249–260, 1995. [Online]. Available: <https://doi.org/10.1007/BF01206331>
- [7] M. Rubinchik and A. M. Shur, “EERTREE: an efficient data structure for processing palindromes in strings,” *Eur. J. Comb.*, vol. 68, pp. 249–265, 2018. [Online]. Available: <https://doi.org/10.1016/j.ejc.2017.07.021>
- [8] T. Mieno, K. Watanabe, Y. Nakashima, S. Inenaga, H. Bannai, and M. Takeda, “Palindromic trees for a

- sliding window and its applications,” *Information Processing Letters*, vol. 173, p. 106174, 2022. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0020019021000892>
- [9] M. Funakoshi and T. Mieno, “Minimal unique palindromic substrings after single-character substitution,” in *String Processing and Information Retrieval*, T. Lecroq and H. Touzet, Eds. Cham: Springer International Publishing, 2021, pp. 33–46.