

题目：基于后缀数组的简单重复序列挖掘算法

实践日期 2022 年 5 月 30 日-6 月 3 日

目 录

一、综合训练目的与要求	1
二、综合训练任务	1
三、问题理解	1
四、设计算法	3
五、描述算法	6
六、证明算法的正确性	8
七、分析算法的时间和空间复杂性	9
八、编码实现算法并测试	10
九、实习日志	11
十、实习总结	13
附录：如核心代码清单等	14

一、综合训练目的与要求

(1) 综合训练目的：

- ①巩固和加深学生对于算法分析与设计课程基本知识的理解和掌握；
- ②培养和利用算法知识解决实际问题的能力；
- ③掌握和利用程序设计语言进行算法程序的开发。调试、测试的能力；
- ④掌握书写算法设计说明文档的能力；
- ⑤提高综合运用算法、程序设计语言、数据结构知识的能力。

(2) 综合训练要求：

- ①学生确认自己的题目，一人一题，班内不可重复。由学委统计题目，并提交给班级的指导老师；
- ②设计算法环节包括选择算法策略、数据结构、设计并描述算法（采用伪代码）、证明算法以及分析算法。需要体现在课程论文中；
- ③实现算法环节包括采用程序设计语言进行编码实现、测试等。需要体现在课程论文中。
- ④答辩材料包括答辩提纲 ppt 和实施计划书；
- ⑤需要提交的材料包括答辩提纲 ppt、实施计划书和完善后的课程论文（包含工作日志等）。

二、综合训练任务

(1) 参照 SA-SSR a suffix array-based algorithm.pdf 论文中构造后缀数组方法，挖掘输入串中的所有串联重复序列。

(2) 读取文件,存储的字符串(测试文件见 test_input.fa)挖掘所有的串联重复序列，并把结果写到文件。

三、问题理解

(3) SSR 序列

简单重复标记序列（Simple Sequence Repeat, SSR）是简单重复序列标记，SSR 是以 PCR 技术为核心的 DNA 分子标记技术，或称微卫星序列标记(Microsatellite sequence, MS)，或短串联重复标记 (Short Tandem Repeat, STR)。SSR 标记具有高度重复性、丰富的多态性、共显性、高度可靠性等优点，但由于其需要对所研究物种的一系列微卫星位点进行克隆和测序分析，以便设计相应的引物，这是非常费时、费力和代价昂贵的工作，没有足够的投资、人力和时间，是不可能实现的，因而给它的利用带来了一定困难。

(4) 后缀数组与最长公共前缀

后缀数组（Suffix Array, SA）主要是两个数组：SA 数组和 rank 数组。其中 $SA[i]$ 表示将所有后缀排序后第 i 小的后缀的编号。 $rank[i]$ 表示后缀 i 的排名。这两个数组满足性质： $sa[rank[i]] = rank[sa[i]] = i$ ，如图 1。

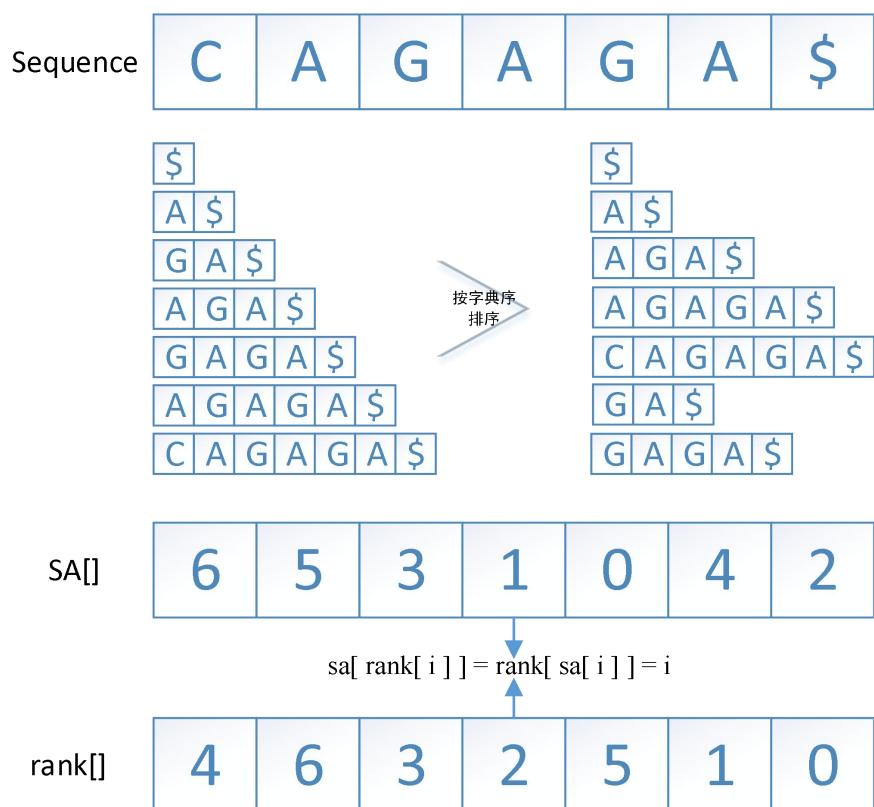


图 1 sequence 的 SA 数组示例

最长公共前缀（Longest Common Prefix，LCP）是指后缀数组中相邻两个后缀的最长公共前缀的长度。在后缀数组的应用中，LCP 是很重要的信息，在实习题目算法介绍论文中定义 LCP(i) 为第 SA[i] 个后缀和第 SA[i-1] 个后缀之间的最长公共前缀长度，如图 2。



图 2 sequence 的 LCP 数组示例

(5) SA-SSR 算法任务

以 AGCT 基因序列作为输入，计算计算基因序列的 SA 数组、LCP 数组，然后根据 SA-SSR（a suffix array-based algorithm for exhaustive and efficient SSR discovery in large genetic sequences，SA-SSR）算法找出输入序列中所有满足要求的 SSR 序列。

四、设计算法

SA-SSR 算法主要涉及三种算法，SA 求解算法，LCP 算法和 SA-SSR 寻找 SSR 算法，下文将对上述三种算法进行介绍。

(1) SA 数组求解算法

常用的后缀数组计算方法有倍增法、DC3 算法，在 SA-SSR 论文中作者使用诱导排序 SA-IS 算法ⁱⁱ。SA-IS 是一种在实际运用中相当快速的、线性时间构建字符串的后缀数组的算法（SA-IS 算法伪代码在第五部分描述算法中给出）。此算法基于诱导排序思想，基本思路是将问题的规模缩小，通过解决更小的问题，获取足够信息，从而快速的解决原始问题。

算法提出了两种后缀类型 L 类型（L_type）和 S 类型（S_type），在论文 Linear Suffix Array Construction by Almost Pure Induced-Sorting 中对 S 类型的定义为“ $S[i]$ is S-type if (i.1) $S[i] < S[i + 1]$ or (i.2) $S[i] = S[i + 1]$ and $\text{suf}(S, i + 1)$ is S-type”；对 L 类型的定义为“ $S[i]$ is L-type if (ii.1) $S[i] > S[i + 1]$ or (ii.2) $S[i] = S[i + 1]$ and $\text{suf}(S, i + 1)$ is L-type.”简单的说，如果 i 位置的后缀的字典序大于 $i+1$ 位置后缀的字典序或者 i 位置后缀是“\$”，那么 i 位置后缀就是 L 类型后缀；如果 i 位置的后缀的字典序小于 $i+1$ 位置后缀的字典序，那么 i 位置后缀就是 S 类型后缀，举例序列 TAGAGA 的类型如图 3。

Sequence	C	A	G	A	G	A	\$
Type	L	S	L	S	L	L	S

图 3 sequence 的后缀类型

算法提出了 LMS（leftmost S-type, LMS）的概念，在论文中定义 LMS 字符为“(LMS character) A character $S[i]$, $i \in [1, n-1]$, is called leftmost S-type (LMS) if $S[i]$ is S-type and $S[i-1]$ is L-type.”；定义 LMS 字符串为“(LMS-substring) A LMS-substring is (i) a substring $S[i..j]$ with both $S[i]$ and $S[j]$ being LMS characters, and there is no other LMS character in the substring, for $i = j$; or (ii) the sentinel itself.”简单的说，LMS 字符一连串的 S 型中最靠左的一个，LMS 字符串就是位置相邻的两个 LMS 字符中间（包括这两个字符）所构成的子串，举例序列 TAGAGA 的 LMS 如图 4。

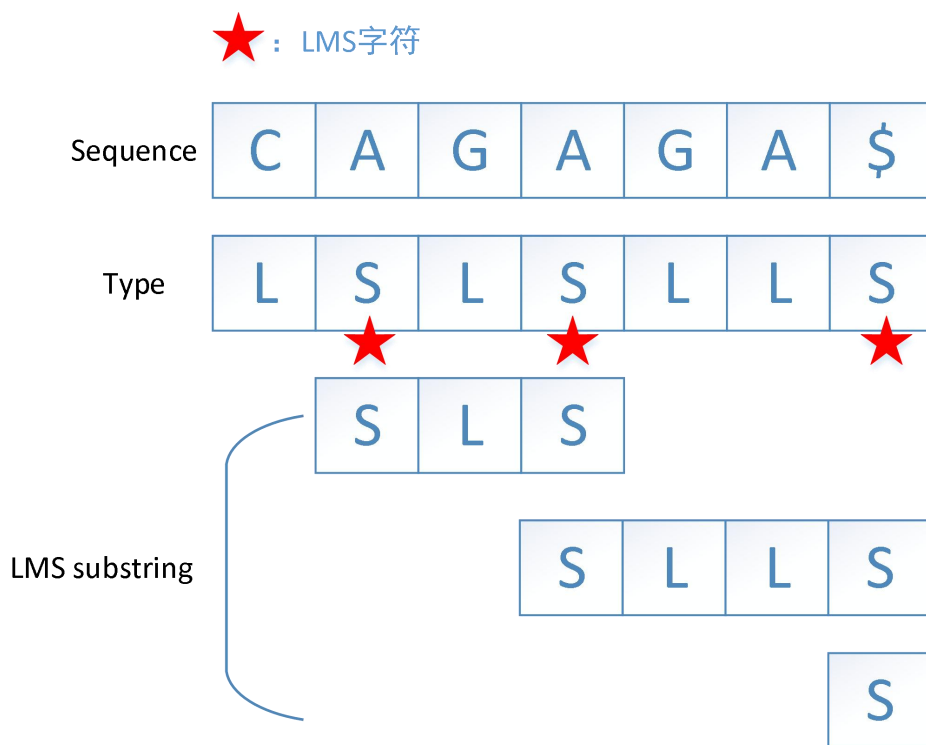


图 4 sequence 的 LMS 串

对 LMS 子串进行排序，将排序后的 LMS 按照字典序进行重新命名，如图 5。如果两个 LMS 字符串相同则使用相同的名称，如果存在同名子串就要递归调用 SA-IS 算法再次计算。



图 5 LMS 重命名

算法进中的诱导排序，从 SA1 诱导至 SA。先确定每个桶的起始位置，确定每个桶 S 型桶的起始位置。将 SA1 中的每一个 * 型后缀按照 SA1 中的顺序放入相应的桶内（如图 6），然后进行诱导排序，排列 L 类型时候从右向左扫描 SA，遇到 L 类型则忽略，继续下一给后缀字符串；排列 S 类型的时候从左向右扫描 SA。这样我们就可以完成从 SA1 诱导到 SA 的排序工作。对于所有的 LMS 后缀，都是有序排放的。从左至右扫描 SA 数组实际上就是按照字典序扫描现有的已排序的后缀。

对于两个相邻的 L 型后缀 A 和 B, 这里假设 $|A| > |B|$, 则必定有 $A > B$ 。由于 B 会被先加入 SA 中, 所以保证了 A 和 B 之间的有序性。又因为 L 型桶是从左往右的顺序加入的, 所以所有的 L 型后缀会逐步地按顺序加入到 SA 中。最后所有的 L 型后缀将会有序。对于 S 型后缀, 除了要注意是相反的顺序和需要重新对 * 型后缀排序外, 其余的原理与 L 型的排序类似。然后对 LMS 类型子串再次进行诱导排序排序, 确定每个桶 S 型桶的起始位置。将每一个 LMS 后缀的首字母 (即 LMS 字符) 按照任意顺序放入对应的桶中, 即可得到最终的 SA 数组。

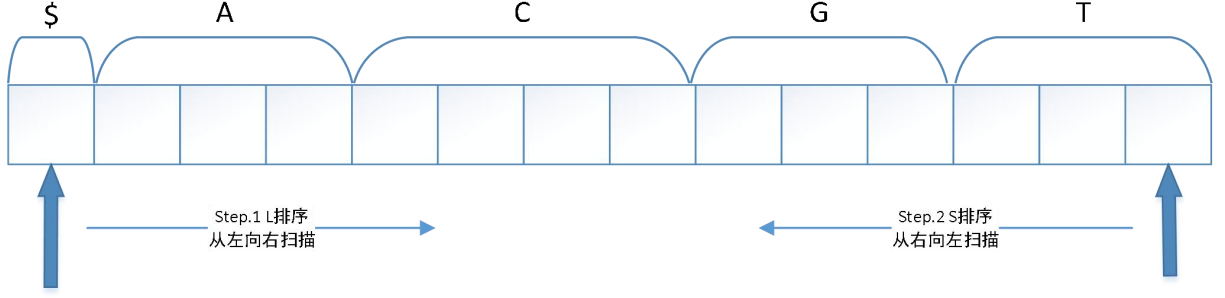


图 6 诱导排序示意图

(2) LCP 求解算法

在 SA-SSR 论文中定义 $LCP(i)$ 为第 $SA[i]$ 个后缀和第 $SA[i-1]$ 个后缀之间的最长公共前缀长度, 如果进行暴力逐个字符从头开始判断, 时间复杂度是巨大的, 严重影响整个算法的时间复杂度。利用 LCP 数组的性质 (公式 1), 可以设计更简单的算法 (LCP 算法伪代码在第五部分描述算法中给出) 计算 LCP 数组。

$$LCP[i] \geq LCP[i-1] - 1 \quad (1)$$

(3) SA-SSR 算法思路

在 SA-SSR 论文中, 作者提出了三个值, 表示 SSR 重复单元长度的 k (公式 2), 表示第一次出现后重复次数的 r (公式 3), 表示第一个周期第一个出现位置的 p (公式 4)。

$$k_i = |SA_i - SA_{i-1}| \quad (2)$$

$$r_i = \left\lfloor \frac{LCP_i}{k_i} \right\rfloor \quad (3)$$

$$p_i = \min(SA_i, SA_{i-1}) \quad (4)$$

根据 SA 数组与 LCP 数组计算 k , r , p 数值 (图 7), 如果 $r > 0$, 则在原始序列的位置 p 处存在长度为 $k * (r + 1)$ 的 SSR, 否则如果 $r = 0$, 则在位置 p 处不存在 SSR。

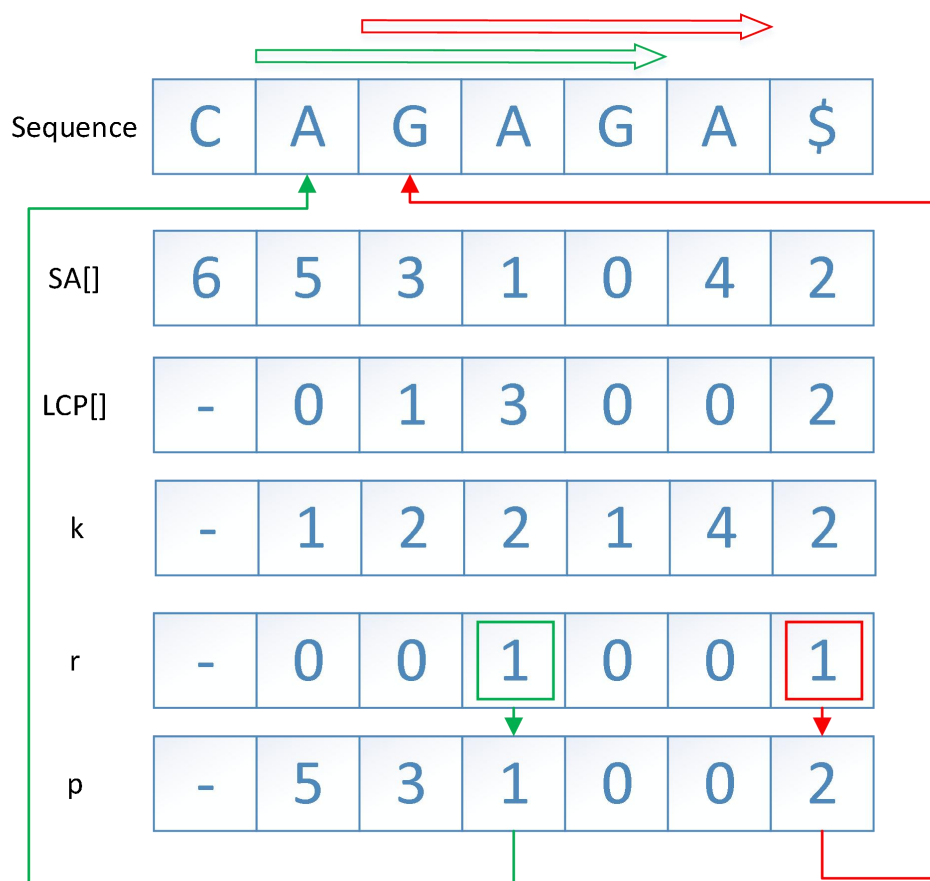


图 7 根据 kpr 寻找 SSR 序列

由此，仅需计算出 sequence 的 SA 数组，根据 SA 数组计算 LCP 数组，然后由 k、p、r 分析序列即可找到输入序列的所有 SSR。

五、描述算法

SA-SSR 算法主要涉及三种算法，SA 求解算法，LCP 算法和 SA-SSR 寻找 SSR 算法，下文将对上述三种算法进行伪代码描述。

(1) SA-IS 算法

function SA-IS(S):

t = bool[]	//标记后缀类型，L 类型/S 类型
S1 = int[]	//重命名后生成的新串
P = int[]	//LMS 标记
bucket = int[]	//桶标记
for i = S.lenth-1 to 0	
t[i] = JudgeType(S,i)	//倒序扫描序列确定后缀类型
for i = t.lenth-1 to 0	
P[i] = JudgeLMS(S,i)	//扫描 t 数组确定所有 LMS
	//对所有 LMS 子串诱导排序
IStoL(LMS)	//先排 L


```

IStoS(LMS)                                //再排 S
                                           //对每一个 LMS 子串重新命名

//生成新的串 S1
S1=RnameLMS()
    if dif(S1)                             //S1 中的每一个字符都不一样
        SA1 = getSA1()                     //直接计算 SA1
    else
        SA1 = SA-IS(S1)                   // 递归计算 SA1
                                           //利用 SA1 诱导排序, 计算 SA
    IStoL(SA1)                             //先排 L
    IStoS(SA1)                             //再排序 S
return SA

```

function LStoL:

```

for i = 0 to SA.lenth

                                           //如果是 L 类型
    if SA[i] !=null && SA[i]>0 && t[SA[i]-1] == L_type
                                           //从左向右依次放入 SA 的桶中
        put SA[i]-1 at the next free slot of the front of t[SA[i] -1]

```

function LStoS:

```

for i = SA.lenth to 0
                                           //如果是 S 类型
    if SA[i] !=null && SA[i]>0 && t[SA[i]-1] == S_type
                                           //从右向左依次放入 SA 的桶中
        put SA[i]-1 at the next free slot of the end of t[SA[i] -1]

```

(2) LCP 算法

function getLCP():

```

    int h =0;                             //前一次计算的两个后缀的 lcp
    LCP[1]=0;
    for i= 1 to sequence.lenth
        int j = SA[rk[i]-1];
        if(h>0) h--;
        while j+h>=n || i+h>=n
            if(sequence.charAt(j+h) != sequence.charAt(i+h)) break;
            h++;                           //如果相同则继续
        LCP[rk[i]] = h;

```

(3) SA-SSR 算法

function SA-SSR:

```

sequence = filetoString()           //读取基因序列为 String 类型
SA=SA-SSR()                         //计算 SA
LCP=getLCP()                        //计算 LCP
for i = 1 to sequence.lenth
    int k = Math.abs(SA[i]-SA[i-1])  //计算 k
    int r = Math.abs((int)LCP[i]/k)  //计算 r
    p=Math.min(SA[i],SA[i-1])        //计算 p
    if(isgoodSSR(r,k))               //如果此 SSR 满足用户需求
        for j = 0 to sequence.lenth
            result+=sequence.charAt(p+j) //记录 SSR 结果
PrintToFile(result)                 // 结果输出到输出文件

```

六、证明算法的正确性

(1) SA-IS 诱导排序正确性证明

①后缀类型判断

判断算法:

如果 $t[i]=S\text{-type}$, 当且仅当下面任意一项成立:

$S[i] < S[i + 1]$

$S[i]=S[i+1]$ 且 $t[i+1]=S\text{-type}$

如果 $t[i]=L\text{-type}$, 当且仅当下面任意一项成立:

$S[i]>S[i+1]$

$S[i]=S[i+1]$ 且 $t[i+1]=L\text{-type}$

正确性证明:

这里证明 S 型的后缀 (L 型是类似的)。对于第一种情况显然是成立的。对于第二种情况, 我们设 $\text{suffix}(S,i)=aA, \text{suffix}(S,i+1)=aB$ 。由于第一个字符是相同的, 因此我们需要比较 A 和 B 的大小。因为它们是连续的后缀, 所以 $A=\text{suffix}(S,i+1), B=\text{suffix}(S,i+2)$ 。由于我们是从右往左推出 t, 所以 A 与 B 的关系实际上可以由 $t[i+1]$ 给出。故 $t[i]=t[i+1]$ 。

②SA1 中两个重命名后缀的字典序关系, 就是 S 中对应的 LMS 型后缀的字典序关系。

证明: 在 SA1 中比较两个字符, 相当于在 S 中比较两个 LMS 子串。由于不存在两个 LMS 子串, 满足其中一个是另外一个子串的真前缀, 所以被比较的两个 LMS 子串要么完全相同, 要么存在一个位置上对应字符不同, 或者在对应位置字符相同时, 对应的后缀类型不同。由于比较 LMS 子串时 S 型的后缀字典序更大, 因此在对应字符相同时, S 型的后缀字典序更大。由此不难得出 S 中的字典序关系和 SA1 中是相同的。

③确定每个桶 S 型桶的起始位置。将 SA1 中的每一个 *型后缀按照 SA1 中的顺序放入相应的桶内。

证明: 对于第二步, 由于放入的 LMS 前缀都只有一个字符, 因为桶的排列是按照字典序的, 所以保证放置后一定有序。

④确定每个桶 L 型桶的起始位置。在 SA 数组中从左往右扫一遍。如果 $SA[i]>0$ 且 $t[SA[i]-1]=L\text{-type}$ ，则将 $SA[i]-1$ 所代表的后缀放入对应的桶中。

证明：当放入第一个 L 型 LMS 前缀时，由于 SA 中还只有 S 型的 LMS 前缀，所以 SA 数组必定是有序的。假设已经放置了 k 个 L 型 LMS 前缀，且它们在 SA 数组中保持有序，现在考虑放入的第 k+1 个 LMS 前缀是否会保证有序。设这个 LMS 前缀为 $pre(S,i)$ ，因为首字母不同的 LMS 前缀之间一定是保持有序的，因此只需考虑它与其首字母相同的 LMS 前缀之间的关系。因为是从左至右扫描使 L 型 LMS 前缀从小至大放置，那么对于所有之前放置的且首字母与其相同的 LMS 前缀 $pre(S,j)$ ，都有 $pre(S,j)<pre(S,i)$ 。假设存在一 LMS 前缀，满足 $pre(S,j)>pre(S,i)$ ，由于 $pre(S,j)[0]=pre(S,i)[0]$ ，所以 $pre(S,j+1)>pre(S,i+1)$ 。而 $pre(S,j+1)$ 与 $pre(S,i+1)$ 一定是之前加入过的（也可能是第二步中的 S 型 LMS 前缀），因此它们之间应当保持有序。而 $pre(S,j)>pre(S,i)$ 告诉我们之前的 SA 数组不是有序的，与假设相反，故不存在这样的 $pre(S,j)$ 。因此，当 $pre(S,i)$ 放置后，SA 数组保持有序。上述归纳直到所有的 L 型 LMS 前缀加入完毕，可以推出所有的 L 型 LMS 前缀之间都是有序的。

(2) SA-SSR 正确性证明

由后缀数组与最长公共前缀的定义以及公式 2 和公式 3 可以得知，如果 $r>0$ ，则 $LCP_i > k_i$ ，即两个排序后相邻后缀之间有重复序列，且此重复序列可能继续延伸，因为 $k_i=SA[i] - SA[i-1]$ ， $LCP_i > k_i$ ，所以在 $p=\min(SA[i],SA[i-1])$ 位置处存在简单重复序列（SSR）。

七、分析算法的时间和空间复杂性

(1) SA-IS 算法时空复杂度

在 SA-IS 算法中，每一步都使用诱导排序将算法时间复杂度降为 $O(n)$ ，但是在算法中存在一个递归调用，由于每次递归都是计算 S1 的后缀数组。因为两个 LMS 字符中间必定有一个 L 型的后缀，所以对于任意的非空 LMS 子串，其长度大于 2，即一个字符串中 LMS 子串的数量不超过 $\lceil |S|/2 \rceil$ 。由此得知

$$|S1| < \lceil |S|/2 \rceil \quad (5)$$

即每一层的递归的问题规模都会减半。因此用公式 6 和公式 7 表示时间复杂度：

$$T(n) = T(\lceil n/2 \rceil) + O(n) \quad (6)$$

SA-IS 时间复杂度为：

$$T(n) = \sum_{k=0}^{\lceil \log n \rceil} 2^k = 2 \times 2^{\lceil \log n \rceil} = O(2^{\lceil \log n \rceil}) = O(n) \quad (7)$$

SA-IS 空间复杂度显然为

$$S(n) = O(n) \quad (8)$$

(2) SA-SSR 时空复杂度

根据论文中的描述 SA-SSR 算法最多需要 $9*n$ (n 是查询序列的长度) 个字节的空间, 构建后缀数组及其最长公共前缀数组的时间复杂度为 $O(n)$, 同时 SA-SSR 算法需要 $3 * (n - 1)$ 次恒定时间计算来找到 SSR, 则有:

SA-SSR 时间复杂度为:

$$T(n) = O(0) \quad (9)$$

SA-SSR 空间复杂度为:

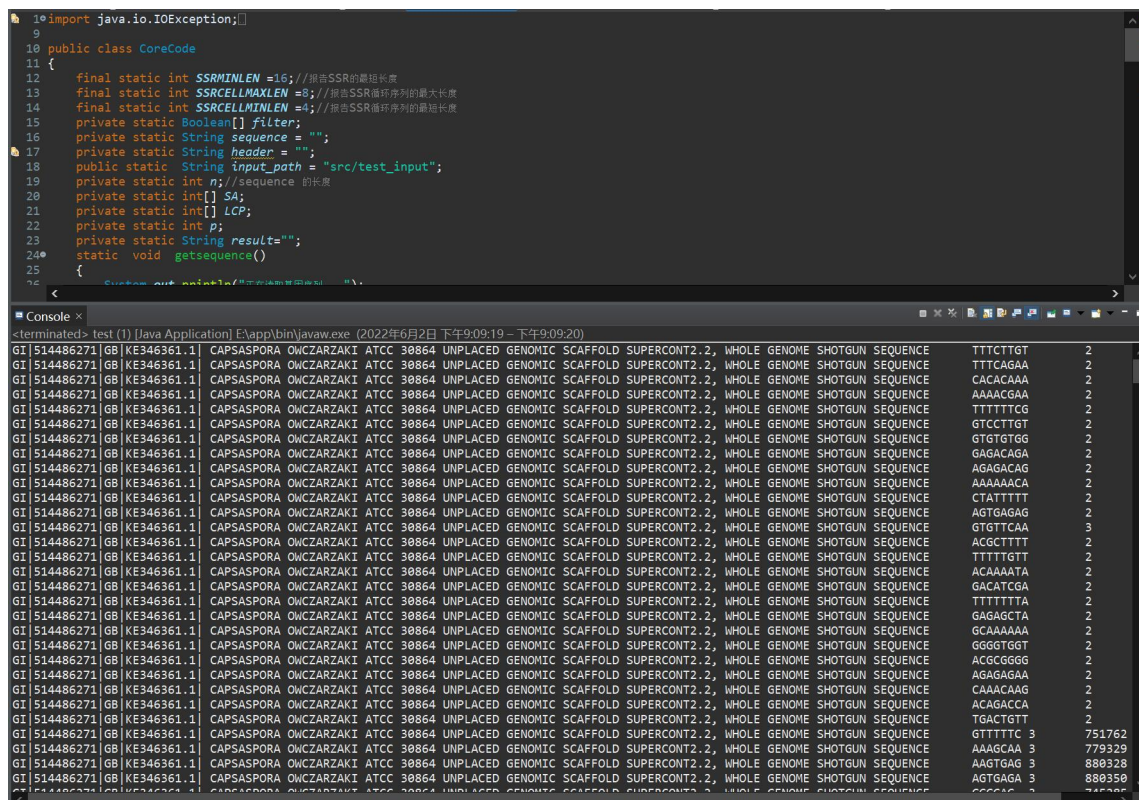
$$S(n) = O(n) \quad (10)$$

八、编码实现算法并测试

(1) 实习代码由 Java 语言实现, 共两个文件, 详见附录。

(2) 测试

相比论文官方代码本文所附代码没有多线程机制, 运行速度较慢。在 CPU 型号为 i7-10510U, RAM 为 16GB, Java 18.9 环境下运行测试 fa 文件时间, 实习代码为 298s, 官方代码为 7s, 测试结果如图 8, 输出文本文件见附件 test_output。



```
import java.io.IOException;

public class CoreCode
{
    final static int SSRMINLEN =16; //报告SSR的最小长度
    final static int SSRCELLMAXLEN =8; //报告SSR循环序列的最大长度
    final static int SSRCELLMINLEN =4; //报告SSR循环序列的最小长度
    private static Boolean[] filter;
    private static String sequence = "";
    private static String header = "";
    public static String input_path = "src/test_input";
    private static int n; //sequence 的长度
    private static int[] SA;
    private static int[] LCP;
    private static int p;
    private static String result="";
    static void getsequence()
    {
        //从文件中读取序列并存储在SA数组中
    }
}

<terminated> test (1) [Java Application] E:\app\bin\javaw.exe (2022年6月2日 下午9:09:19 - 下午9:09:20)
GI [514486271] GB [KE346361.1] CAPSASPORA OWCZARZAKI ATCC 30864 UNPLACED GENOMIC SCAFFOLD SUPERCONT2.2, WHOLE GENOME SHOTGUN SEQUENCE TTTCTTGT 2
GI [514486271] GB [KE346361.1] CAPSASPORA OWCZARZAKI ATCC 30864 UNPLACED GENOMIC SCAFFOLD SUPERCONT2.2, WHOLE GENOME SHOTGUN SEQUENCE TTTTCAGAA 2
GI [514486271] GB [KE346361.1] CAPSASPORA OWCZARZAKI ATCC 30864 UNPLACED GENOMIC SCAFFOLD SUPERCONT2.2, WHOLE GENOME SHOTGUN SEQUENCE CACACAGAA 2
GI [514486271] GB [KE346361.1] CAPSASPORA OWCZARZAKI ATCC 30864 UNPLACED GENOMIC SCAFFOLD SUPERCONT2.2, WHOLE GENOME SHOTGUN SEQUENCE AAAACAGAA 2
GI [514486271] GB [KE346361.1] CAPSASPORA OWCZARZAKI ATCC 30864 UNPLACED GENOMIC SCAFFOLD SUPERCONT2.2, WHOLE GENOME SHOTGUN SEQUENCE TTTTTCGT 2
GI [514486271] GB [KE346361.1] CAPSASPORA OWCZARZAKI ATCC 30864 UNPLACED GENOMIC SCAFFOLD SUPERCONT2.2, WHOLE GENOME SHOTGUN SEQUENCE GTCCTTGT 2
GI [514486271] GB [KE346361.1] CAPSASPORA OWCZARZAKI ATCC 30864 UNPLACED GENOMIC SCAFFOLD SUPERCONT2.2, WHOLE GENOME SHOTGUN SEQUENCE GTGTGTGT 2
GI [514486271] GB [KE346361.1] CAPSASPORA OWCZARZAKI ATCC 30864 UNPLACED GENOMIC SCAFFOLD SUPERCONT2.2, WHOLE GENOME SHOTGUN SEQUENCE GAGACAGAA 2
GI [514486271] GB [KE346361.1] CAPSASPORA OWCZARZAKI ATCC 30864 UNPLACED GENOMIC SCAFFOLD SUPERCONT2.2, WHOLE GENOME SHOTGUN SEQUENCE AGAGACAG 2
GI [514486271] GB [KE346361.1] CAPSASPORA OWCZARZAKI ATCC 30864 UNPLACED GENOMIC SCAFFOLD SUPERCONT2.2, WHOLE GENOME SHOTGUN SEQUENCE AAAAGAGAA 2
GI [514486271] GB [KE346361.1] CAPSASPORA OWCZARZAKI ATCC 30864 UNPLACED GENOMIC SCAFFOLD SUPERCONT2.2, WHOLE GENOME SHOTGUN SEQUENCE CTATTTTT 2
GI [514486271] GB [KE346361.1] CAPSASPORA OWCZARZAKI ATCC 30864 UNPLACED GENOMIC SCAFFOLD SUPERCONT2.2, WHOLE GENOME SHOTGUN SEQUENCE AGTGAGAG 2
GI [514486271] GB [KE346361.1] CAPSASPORA OWCZARZAKI ATCC 30864 UNPLACED GENOMIC SCAFFOLD SUPERCONT2.2, WHOLE GENOME SHOTGUN SEQUENCE GTGTTCAA 3
GI [514486271] GB [KE346361.1] CAPSASPORA OWCZARZAKI ATCC 30864 UNPLACED GENOMIC SCAFFOLD SUPERCONT2.2, WHOLE GENOME SHOTGUN SEQUENCE ACCTTTTT 2
GI [514486271] GB [KE346361.1] CAPSASPORA OWCZARZAKI ATCC 30864 UNPLACED GENOMIC SCAFFOLD SUPERCONT2.2, WHOLE GENOME SHOTGUN SEQUENCE TTTTGTGT 2
GI [514486271] GB [KE346361.1] CAPSASPORA OWCZARZAKI ATCC 30864 UNPLACED GENOMIC SCAFFOLD SUPERCONT2.2, WHOLE GENOME SHOTGUN SEQUENCE ACAAAATA 2
GI [514486271] GB [KE346361.1] CAPSASPORA OWCZARZAKI ATCC 30864 UNPLACED GENOMIC SCAFFOLD SUPERCONT2.2, WHOLE GENOME SHOTGUN SEQUENCE GACATCGA 2
GI [514486271] GB [KE346361.1] CAPSASPORA OWCZARZAKI ATCC 30864 UNPLACED GENOMIC SCAFFOLD SUPERCONT2.2, WHOLE GENOME SHOTGUN SEQUENCE TTTTTTTA 2
GI [514486271] GB [KE346361.1] CAPSASPORA OWCZARZAKI ATCC 30864 UNPLACED GENOMIC SCAFFOLD SUPERCONT2.2, WHOLE GENOME SHOTGUN SEQUENCE GAGAGCTA 2
GI [514486271] GB [KE346361.1] CAPSASPORA OWCZARZAKI ATCC 30864 UNPLACED GENOMIC SCAFFOLD SUPERCONT2.2, WHOLE GENOME SHOTGUN SEQUENCE GCAGAGAG 2
GI [514486271] GB [KE346361.1] CAPSASPORA OWCZARZAKI ATCC 30864 UNPLACED GENOMIC SCAFFOLD SUPERCONT2.2, WHOLE GENOME SHOTGUN SEQUENCE GGGGTGGT 2
GI [514486271] GB [KE346361.1] CAPSASPORA OWCZARZAKI ATCC 30864 UNPLACED GENOMIC SCAFFOLD SUPERCONT2.2, WHOLE GENOME SHOTGUN SEQUENCE ACCCGGGG 2
GI [514486271] GB [KE346361.1] CAPSASPORA OWCZARZAKI ATCC 30864 UNPLACED GENOMIC SCAFFOLD SUPERCONT2.2, WHOLE GENOME SHOTGUN SEQUENCE AGAGAGAA 2
GI [514486271] GB [KE346361.1] CAPSASPORA OWCZARZAKI ATCC 30864 UNPLACED GENOMIC SCAFFOLD SUPERCONT2.2, WHOLE GENOME SHOTGUN SEQUENCE CACAAGAG 2
GI [514486271] GB [KE346361.1] CAPSASPORA OWCZARZAKI ATCC 30864 UNPLACED GENOMIC SCAFFOLD SUPERCONT2.2, WHOLE GENOME SHOTGUN SEQUENCE ACAGACCA 2
GI [514486271] GB [KE346361.1] CAPSASPORA OWCZARZAKI ATCC 30864 UNPLACED GENOMIC SCAFFOLD SUPERCONT2.2, WHOLE GENOME SHOTGUN SEQUENCE TGACTGTT 2
GI [514486271] GB [KE346361.1] CAPSASPORA OWCZARZAKI ATCC 30864 UNPLACED GENOMIC SCAFFOLD SUPERCONT2.2, WHOLE GENOME SHOTGUN SEQUENCE GTTTTTC 3 751762
GI [514486271] GB [KE346361.1] CAPSASPORA OWCZARZAKI ATCC 30864 UNPLACED GENOMIC SCAFFOLD SUPERCONT2.2, WHOLE GENOME SHOTGUN SEQUENCE AAAGCAA 3 779329
GI [514486271] GB [KE346361.1] CAPSASPORA OWCZARZAKI ATCC 30864 UNPLACED GENOMIC SCAFFOLD SUPERCONT2.2, WHOLE GENOME SHOTGUN SEQUENCE AAGTGAG 3 880328
GI [514486271] GB [KE346361.1] CAPSASPORA OWCZARZAKI ATCC 30864 UNPLACED GENOMIC SCAFFOLD SUPERCONT2.2, WHOLE GENOME SHOTGUN SEQUENCE AGTGAGA 3 880350
GI [514486271] GB [KE346361.1] CAPSASPORA OWCZARZAKI ATCC 30864 UNPLACED GENOMIC SCAFFOLD SUPERCONT2.2, WHOLE GENOME SHOTGUN SEQUENCE CCGGAG 3 745385
```

图 8 实习代码测试截图

九、实习日志

5月30日 星期一 上午

原定任务：选题，明确题目要求

实习进度：

选题，基于后缀数组的简单重复序列挖掘算法（SA-SSR 算法）

题目要求，阅读论文 SA-SSR: a suffix array-based algorithm for exhaustive and efficient SSR discovery in large genetic sequences，实现论文中的 SA-SSR 的算法，挖掘老师提供的测试数据（test_input.fa）中的简单重复序列（SSR），将其结果写入输出文件中。

5月30日 星期一 下午

原定任务：设计算法

实习进度：确定选题之后便开始阅读论文 SA-SSR: a suffix array-based algorithm for exhaustive and efficient SSR discovery in large genetic sequences。论文是全英版，初次接触英文版本的学术文献还是有些不适应，配合翻译工具，一点一点的将论文读通，基本掌握了论文描述算法的主要思想。

5月31日 星期二 上午

原定任务：设计算法

实习进度：论文中指出官方代码在 github 上公开，所以去 github 将论文官方代码下载开始阅读官方代码。由于 github 网站的 DNS 污染问题，登陆 github 下载源码花费了不少时间。代码下载之后，在 linux 环境下调试并运行了一下，9 万行数据在几秒钟的时间里便计算完成开始输出，感觉这 SA-SSR 算法还是有点东西。

5月31日 星期二 下午

原定任务：设计算法

实习进度：官方代码给的确实有点多，上午基本都在调试官方代码，还没有读几行代码便到了下课时间了。所以鞋屋继续阅读官方代码。经过几个小时的研读，发现官方代码中使用了多线程机制和生产者消费者机制，而且计算后缀数组的算法是 SA-IS 诱导排序算法，研究了一下午也没有研究明白这个 SA-IS 算法的意思。而且在网上也没找到 SA-IS 相关的关键字，这让我很是头大，这个算法看代码也看不懂，搜也搜不到，实习进度似乎陷入了瓶颈。

6月1日 星期三 上午

原定任务：实现算法

实习进度：昨天下午研究了一下午官方代码里的后缀数组构建算法，没有丝毫头绪，所以昨晚又加班研究了一下，这次换了一个思路，既然是构建后缀数组，那么检索后缀数组的计算方法总可以了吧。只要最后能够计算出后缀数组，那么 SA-SSR 算法的实现也就迎刃而解。上午找了许多计算后缀数组的算法，包括 DC3 算法，倍增法，当然还有暴力法。在看一篇博文的时候发现博主提到了一个算法 Induce Sort，叫诱导排序是中山大学农革教授的一篇论文，据说是迄今为止构建后缀数组的性能最快的算法。我看到他的那篇论文题目 Linear Suffix Array Construction by Almost Pure Induced-Sorting，我才恍然大悟，原来 SA-SSR 论文官方代码使用的后缀数组构建方法是诱导排序算法。

6月1日 星期三 下午

原定任务：实现算法

实习进度：上午终于突破了瓶颈，找到了实现的方向。下午便开始阅读农革的论文以及其他诱导排序相关的文献。不得不说，虽然诱导排序性能表现确实好，但是思想是真的难懂，算法流程是真的复杂。一整个下午都在学习 SA-IS 算法。

6月1日 星期三 晚上

实习进度：由于前面的时间耽搁，实习进度一下落后了，所以晚上不得不加班赶一下进度了。当然也是因为下午刚刚学习了 SA-IS 算法，手有些痒，想抓紧动手实现一下，所以晚上加班实现了一下 SA-IS 算法的代码。不得不说 SA-IS 代码还是有些晦涩难懂的，虽然有 C++ 的代码对照，也有很多文章对 IS 算法介绍的更通俗，但是我还是花了 4 个多小时的时间才将 SA-IS 算法用 Java 复现出来，在阶梯教室从 7 点待到 11 点多才实现了 SA-IS 算法。主要是 SA-IS 算法中对 LMS 重命名然后再次递归诱导排序部分确实是不好懂，代码实现起来太绕了。然后后面实现 LCP 数组构建算法和 SA-SSR 算法都是比较简单的。注意在构建 LCP 数组的时候要记得用公式 1 优化计算，否则 LCP 计算的时间复杂度太大了，严重影响算法时间。在 LCP 与 SA-SSR 算法实现花了 1 个多小时。也就是说在 6 月 2 日凌晨 1 点的时候我的代码终于可以跑了！

6月2日 星期四 上午

原定任务：完善算法

实习进度：昨晚调试了一晚上代码终于可以跑了，但是今天早上对比官方代码的运行结果时发现我的代码跑出来与官方代码有出入。原来是我没有实现 isgoogSSR 算法筛选 SSR 序列，而是将找到的 SSR 全部放出来。官方代码默认只查找总长度大于 16 的序列，且重叠序列仅报告最左边的序列，封闭 SSR 仅报告最长的序列，仅报告重复次数大于 2 的序列。由此加入 isgoodSSR 方法进行筛选之后，代码运行结果与官方代码运行结果完全一致！但是毕竟我没有在我的代码中加入多线程机制，所以美中不足的是我的单线程跑起来比官方代码慢太多了。本来想是否可以加一下多线程，再完善一下代码，但是时间确实不多了，今天已经是第四天了，第五天就要答辩了，而且多线程也不是这 SA-SSR 算法的主要部分，不实现多线程应该影响不大吧，毕竟是算法实习，不是操作系统实习。

6月2日 星期四 下午

原定任务：准备答辩材料

实习进度：下午写了一下午实习报告，已经写了六千字了，虽然我觉得描述的还不够全面，但是时间实在是来不及了。看来今晚又要加班写答辩 PPT 了。

6月2日 星期四 晚上

实习进度：把下午没有写完的实习报告写完，然后开始准备答辩 PPT。

十、实习总结

（1）任务总结

本次实习用 Java 实现了 SA-SSR 论文中的 SA-SSR 算法，实习代码能够完成 SSR 序列的搜索任务，测试输入样例在本人计算机上 298s 可以运行完毕给出正确结果。

（2）存在不足

由于个人水平有限，且实习时间不充裕，本次实习代码没有加入多线程机制，相比官方代码运行速度较慢。

（3）收获与反思

收获：通过本次实习，第一次接触了算法领域的学术论文，并深入研究与实现了论文提出的算法。以本次实习为契机学习了后缀数组的构建算法，同时也复习了 java 语言编程内容，可谓收获颇深，本次实习经历在以后的学习与生活中将是一笔宝贵的财富！

反思：通过本次实习暴露出了个人许多问题。首先是逻辑思维与代码能力的问题，本文的算法思想还是比较简单的，但是实习过程中实现算法的时候却花费了大量的时间，个人认为是缺少练习且对编程语言不熟悉的缘故，因此在后面的学习中至少要注重代码能力与逻辑思维能力的锻炼与提高。恰逢暑期，可以在暑假期间多做几道算法题来提升自己。同样重要就是英文阅读能力的问题，实习过程中阅读英文文献确实存在障碍，况且本人有读研的打算，如果英文阅读能力不提升，以后的学习生涯定会对出几道坎坷。

最后感谢老师在实习过程中的悉心指导与帮助！

附录：核心代码清单

CoreCode.java:

```
import java.io.FileWriter;
import java.io.IOException;
import java.nio.charset.Charset;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Queue;

public class CoreCode
{
    final static int SSRMINLEN =16;//报告 SSR 的最短长度
    final static int SSRCELLMAXLEN =8;//报告 SSR 循环序列的最大长度
    final static int SSRCELLMINLEN =4;//报告 SSR 循环序列的最短长度
    private static Boolean[] filter;
    private static String sequence = "";
    private static String header = "";
    public static String input_path = "src/test_input";
    private static int n;//sequence 的长度
    private static int[] SA;
    private static int[] LCP;
    private static int p;
    private static String result="";
    private static Queue<String> sequenceque = new LinkedList<>();//基因序列队列
    private static Queue<String> headque = new LinkedList<>();//基因序列名队列
    private static String outputPath = "./test_output.txt";
    private static void saveAsFileWriter(String content) {
        FileWriter fwriter = null;
        try {
            fwriter = new FileWriter(outputPath, true);
            fwriter.write(content);
        } catch (IOException ex) {
            ex.printStackTrace();
        } finally {
            try {
                fwriter.flush();
                fwriter.close();
            }
        }
    }
}
```



```

        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}
static void getsequence()
{
    sequence=null;
    System.out.println("正在读取基因序列...");
    try {
        List<String> lines = Files.readAllLines(Paths.get(input_path),Charset.forName("UTF-8"));
        for(int i=0;i<lines.size();++i)
        {
            if(lines.get(i).charAt(0) == '>')
            {
                if(sequence != null) sequencequeue.add(sequence);
                lines.get(i).charAt(0) == ";
                header = lines.get(i).toString();
                headque.add(header);
                i++;
            }
            else
            {
                sequence += lines.get(i).toString().toUpperCase();
            }
        }
        if(sequence != null)
        {
            sequencequeue.offer(sequence);
        }
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

    n = sequence.length();
    filter = new Boolean[n+1];
    LCP = new int[n+1];
    System.out.println("读入完成");

}

```

#####-getLC

P

```

static void getLCP()
{
    int[] rk = new int[n+1];
    for(int i=1;i<=n;++i)
    {
        rk[SA[i]] = i;
    }
    int h=0;//前一次计算的两个后缀的 lcp
    LCP = new int[n+1];
    LCP[1]=0;
    for(int i=1;i<n;++i)
    {
        int j = SA[rk[i]-1];
        if(h>0) h--;
        for(;j+h<n && i+h<n;h++)
        {
            if(sequence.charAt(j+h) != sequence.charAt(i+h)) break;
        }
        LCP[rk[i]] = h;
    }
}

#####-getLCP
P

#####-isgood
SSR

static boolean isgoodSSR(int r,int k)//SSR 过滤
{
    int len = k*(r+1);
    if(r<=0) return false;
    else if(len<SSRMINLEN) return false;
    else if (k>SSRCELLMAXLEN) return false;
    else if (k<SSRCELLMINLEN) return false;
    for(int i=0;i<len;++i)
    {
        if(filter[p+i] != null)
        {
            return false;
        }
        filter[p+i]=true;
    }
    return true;
}

#####-isgood

```

SSR

```
public static void main(String[] args) {
    getsequence();
    //-测试
    // sequence ="CAGAGAS$";
    // n = sequence.length();
    // filter = new Boolean[n+1];
    // LCP = new int[n+1];
    //-测试
    System.out.println(sequence.size());
    for(int se=0;se<headque.size();++se)
    {
        sequence = sequence.remove();
        header = headque.remove();
        System.out.println(sequence);
        result=header;
        SAIS sais = new SAIS();
        SA = sais.makeSuffixArrayByInducedSorting(sequence,256);
        System.out.println("第"+se+"个序列: "+header+" 开始查找");
        System.out.println("SA 数组计算完成");
        getLCP();
        System.out.println("LCP 数组计算完成");
        System.out.println("正在查找 SSR 序列...");
        saveAsFileWriter("SSR\t\trepest\t\tposition\n");
        //System.out.println("SSR\t\trepest\t\tposition\n");
        for(int i = 2;i<=n;++i)
        {
            int k = Math.abs(SA[i]-SA[i-1]);
            int r = Math.abs((int)LCP[i]/k);
            p=Math.min(SA[i],SA[i-1]);
            if(isgoodSSR(r,k))
            //if(r>0)
            {
                for(int j=0;j<k;++j)
                {
                    result+=sequence.charAt(p+j);
                }
                result+="\t\t";
                r++;
                result+=r;
                result+="\t\t";
                result+=p;
                result+="\n";
            }
        }
    }
}
```

```

        result+=header;
    }
}
saveAsFileWriter(result);
System.out.println("查找完毕，结果已经写入文件"+outputPath);
}
}

```

SAIS.java:

```

import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;

public class SAIS {

    public char[] buildTypeMap(String input){
        char inp[] = input.toCharArray();
        int length = input.length();
        char result[] = new char[length + 1];
        result[length] = 'S';
        result[length - 1] = 'L';
        for(int i=length-2; i>=0; i--){
            if( inp[i] > inp[i+1] ){
                result[i] = 'L';
            }
            else if( inp[i] == inp[i+1] && result[i+1] == 'L'){
                result[i] = 'L';
            }
            else{
                result[i] = 'S';
            }
        }
        return result;
    }

    public boolean isLMSChar(int offset, char[] typemap){

        if(offset == 0){
            return false;
        }
        if(typemap[offset] == 'S' && typemap[offset-1] == 'L'){
            return true;
        }
    }
}

```

```

    }
    return false;
}

public boolean lmsSubstringsAreEqual(String string, char[] typemap, int offsetA, int offsetB){
    if(offsetA == string.length() || offsetB == string.length()){
        return false;
    }
    int i = 0;
    char[] str = string.toCharArray();
    while(true){
        boolean aIsLMS = isLMSChar(i+offsetA, typemap);
        boolean bIsLMS = isLMSChar(i+offsetB, typemap);
        if(i>0 && aIsLMS && bIsLMS){
            return true;
        }
        if(aIsLMS != bIsLMS){
            return false;
        }
        if(str[i+offsetA] != str[i+offsetB]){
            return false;
        }
        i++;
    }
}

public int[] findBucketSizes(String string, int alphabetSize){
    int[] res = new int[alphabetSize];
    Arrays.fill(res, 0);
    char[] str = string.toCharArray();
    for(char c: str){
        res[c] += 1;
    }
    return res;
}

public int[] findBucketHeads(int[] bucketSizes){
    int offset = 1;
    int[] res = new int[bucketSizes.length];
    for(int i=0; i<res.length; i++){
        res[i] = offset;
        offset += bucketSizes[i];
    }
    return res;
}

```

```

    }

    public int[] findBucketTailss(int[] bucketSizes){
        int offset = 1;
        int[] res = new int[bucketSizes.length];
        for(int i=0;i<res.length;i++){
            offset += bucketSizes[i];
            res[i] = offset - 1;
        }
        return res;
    }

    public int[] makeSuffixArrayByInducedSorting(String string, int alphabetSize)
    {
        char[] typemap = buildTypeMap(string);
        int[] bucketSizes = findBucketSizes(string, alphabetSize);
        int[] guessedSuffixArray = guessLMSSort(string, bucketSizes, typemap);

        guessedSuffixArray = induceSortL(string, guessedSuffixArray, bucketSizes, typemap);
        guessedSuffixArray = induceSortS(string, guessedSuffixArray, bucketSizes, typemap);

        HashMap<Integer,ArrayList<Integer>> res = summariseSuffixArray(string, guessedSuffixArray,
typemap);
        ArrayList<Integer> summarySuffixOffsets = res.get(0);
        ArrayList<Integer> summaryString = res.get(1);

        int summaryAlphabetSize = res.get(2).get(0);
        int[] summarySuffixOffsetsArray = new int[summarySuffixOffsets.size()];
        int[] summaryStringArray = new int[summaryString.size()];

        for (int i=0; i < summarySuffixOffsetsArray.length; i++)
        {
            summarySuffixOffsetsArray[i] = summarySuffixOffsets.get(i).intValue();
            summaryStringArray[i] = summaryString.get(i).intValue();
        }

        int[] summarySuffixArray = makeSummarySuffixArray(summaryStringArray,
summaryAlphabetSize);
        int[] result = accurateLMSSort(string, bucketSizes, typemap, summarySuffixArray,
summarySuffixOffsetsArray);

        result = induceSortL(string, result, bucketSizes, typemap);
        result = induceSortS(string, result, bucketSizes, typemap);
    }

```

```

        return result;
    }

    public int[] accurateLMSSort(String string, int[] bucketSizes, char[] typemap, int[]
summarySuffixArray,
        int[] summarySuffixOffsets) {
        int[] suffixOffsets = new int[string.length() + 1];
        Arrays.fill(suffixOffsets, -1);
        int[] bucketTails = findBucketTailss(bucketSizes);
        for(int i=summarySuffixArray.length-1;i>1;i--){
            int stringIndex = summarySuffixOffsets[summarySuffixArray[i]];
            int bucketIndex = string.charAt(stringIndex);
            suffixOffsets[bucketTails[bucketIndex]] = stringIndex;
            bucketTails[bucketIndex] -= 1;
        }
        suffixOffsets[0] = string.length();
        return suffixOffsets;
    }

    public int[] makeSummarySuffixArray(int[] summaryString, int summaryAlphabetSize) {
        int[] summarySuffixArray;
        if(summaryAlphabetSize == summaryString.length){
            summarySuffixArray = new int[summaryString.length + 1];
            summarySuffixArray[0] = summaryString.length;
            for(int i=0;i<summaryString.length;i++){
                int y = summaryString[i];
                summarySuffixArray[y+1] = i;
            }
        }
        else{
            StringBuilder s = new StringBuilder();
            for(int i=0;i<summaryString.length;i++){
                char c = (char) summaryString[i];
                s.append(c);
            }
            summarySuffixArray = makeSuffixArrayByInducedSorting(s.toString(),
summaryAlphabetSize);
        }
        return summarySuffixArray;
    }

    public int[] induceSortL(String string, int[] guessedSuffixArray, int[] bucketSizes, char[] typemap)
    {

```

```

    int[] bucketHeads = findBucketHeads(bucketSizes);
    for(int i=0;i<guessedSuffixArray.length;i++){
        if(guessedSuffixArray[i] == -1){
            continue;
        }
        int j = guessedSuffixArray[i] - 1;
        if(j<0){
            continue;
        }
        if(typemap[j] != 'L'){
            continue;
        }
        int bucketIndex = string.charAt(j);
        guessedSuffixArray[bucketHeads[bucketIndex]] = j;
        bucketHeads[bucketIndex] += 1;
    }
    return guessedSuffixArray;
}

public int[] induceSortS(String string, int[] guessedSuffixArray, int[] bucketSizes, char[] typemap) {
    int[] bucketTails = findBucketTailss(bucketSizes);
    for(int i = guessedSuffixArray.length-1;i>-1;i--){
        int j = guessedSuffixArray[i] - 1;
        if(j<0){
            continue;
        }
        if(typemap[j] != 'S'){
            continue;
        }
        int bucketIndex = string.charAt(j);
        guessedSuffixArray[bucketTails[bucketIndex]] = j;
        bucketTails[bucketIndex] -= 1;
    }
    return guessedSuffixArray;
}

public int[] guessLMSSort(String string, int[] bucketSizes, char[] typemap) {
    int[] guessedSuffixArray = new int[string.length() + 1];
    Arrays.fill(guessedSuffixArray, -1);
    int[] bucketTails = findBucketTailss(bucketSizes);
    for(int i=0;i<string.length();i++){
        if(!isLMSSChar(i, typemap)){
            continue;
        }
    }
}

```



```

        int bucketIndex = string.charAt(i);
        guessedSuffixArray[bucketTails[bucketIndex]] = i;
        bucketTails[bucketIndex] -= 1;
    }
    guessedSuffixArray[0] = string.length();

    return guessedSuffixArray;
}

public HashMap<Integer, ArrayList<Integer>> summariseSuffixArray(String string, int[]
guessedSuffixArray, char[] typemap){
    int[] lmsNames = new int[string.length() + 1];
    Arrays.fill(lmsNames, -1);
    int currentName = 0;
    int lastLMSSuffixOffset;
    lmsNames[guessedSuffixArray[0]] = currentName;
    lastLMSSuffixOffset = guessedSuffixArray[0];
    for(int i = 1; i < guessedSuffixArray.length; i++){
        int suffixOffset = guessedSuffixArray[i];
        if(!isLMSChar(suffixOffset, typemap)){
            continue;
        }
        if(!lmsSubstringsAreEqual(string, typemap, lastLMSSuffixOffset, suffixOffset)){
            currentName += 1;
        }
        lastLMSSuffixOffset = suffixOffset;
        lmsNames[suffixOffset] = currentName;
    }
    ArrayList<Integer> summarySuffixOffsets = new ArrayList<>();
    ArrayList<Integer> summaryString = new ArrayList<>();
    for(int i = 0; i < lmsNames.length; i++){
        int name = lmsNames[i];
        if(name == -1){
            continue;
        }
        summarySuffixOffsets.add(i);
        summaryString.add(name);
    }
    ArrayList<Integer> summaryAlphabetSize = new ArrayList<>();
    summaryAlphabetSize.add(currentName + 1);

    HashMap<Integer, ArrayList<Integer>> res = new HashMap<>();
    res.put(0, summarySuffixOffsets);
    res.put(1, summaryString);
}

```

```

        res.put(2, summaryAlphabetSize);

    return res;

}

}

```

ⁱ Pickett, B. D., Karlinsey, S. M., Penrod, C. E., Cormier, M. J., Ebbert, M. T. W., Shiozawa, D. K., Whipple, C. J., & Ridge, P. G. (2016). SA-SSR: a suffix array-based algorithm for exhaustive and efficient SSR discovery in large genetic sequences: Table 1. *Bioinformatics*, 32(17), 2707–2709. <https://doi.org/10.1093/bioinformatics/btw298>

ⁱⁱ Nong, G., Zhang, S., & Chan, W. H. (2009, March). Linear Suffix Array Construction by Almost Pure Induced-Sorting. 2009 Data Compression Conference. <https://doi.org/10.1109/dcc.2009.42>