处理字符串的有力工具 ---

后缀数组

华南师范大学附属中学 罗穗骞

指导教师：张学东

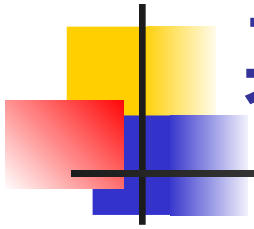
目录

- 第一部分 后缀数组的实现
- 　DC3 算法

- 第二部分 后缀数组的应用
- 　例 1：求两个字符串的最长公共子串

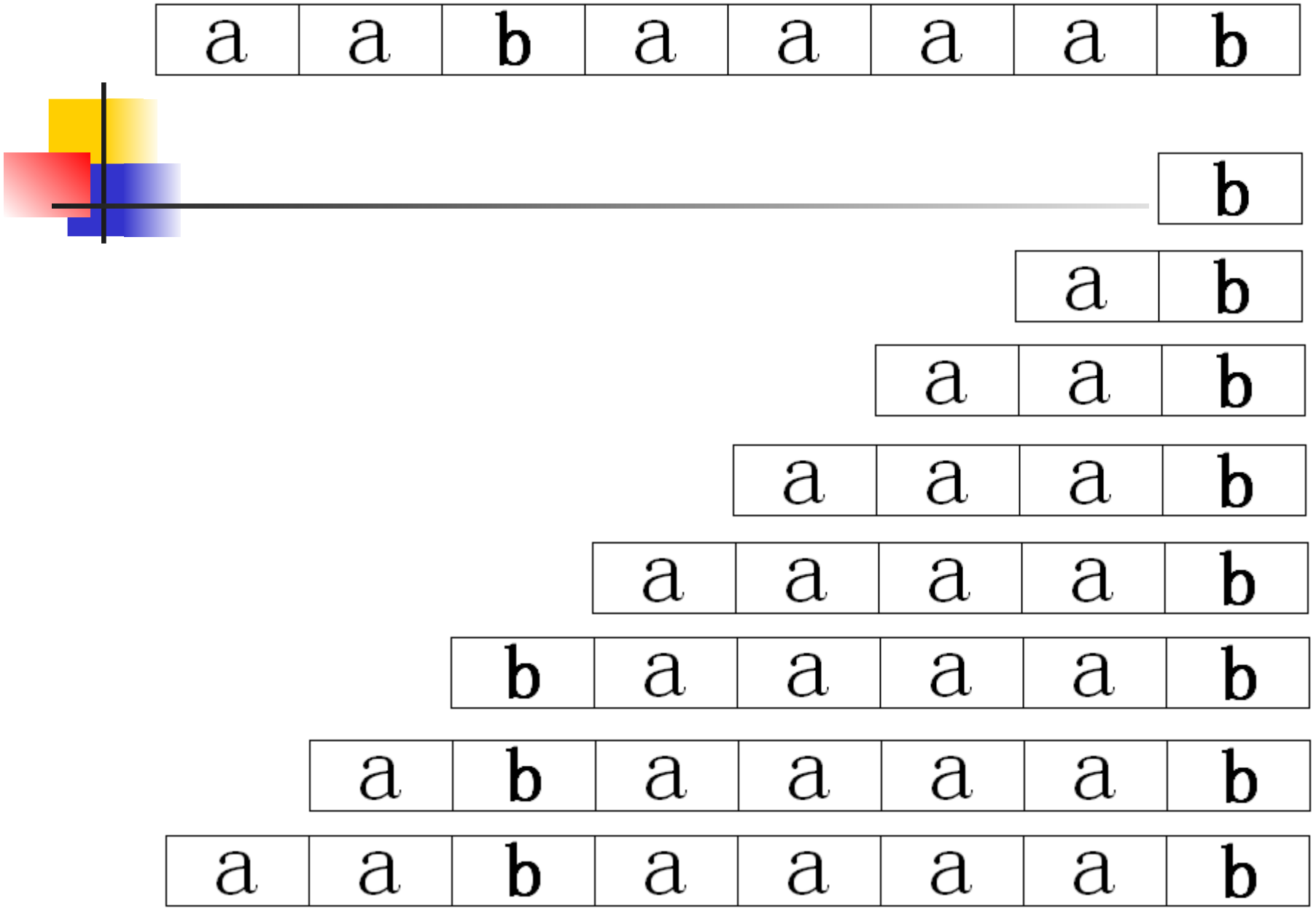
基本定义

- 【后缀数组 SA 】后缀数组保存的是一个字符串的所有后缀的排序结果。其中 SA[i] 保存的是字符串所有的后缀中第 i 小的后缀的开头位置。
- 【名次数组 Rank 】名次数组 Rank[i] 保存的是后缀 i 在所有后缀中从小到大排列的“名次”。
- 为了叙述方便，以第 k 个字符开始的后缀称为后缀 k 。



基本定义

- 以字符串“ aabaaaab” 为例。


a a b a a a a b
b
a b
a a b
a a a b
a a a a b
b a a a a b
a b a a a a b
a a b a a a a b

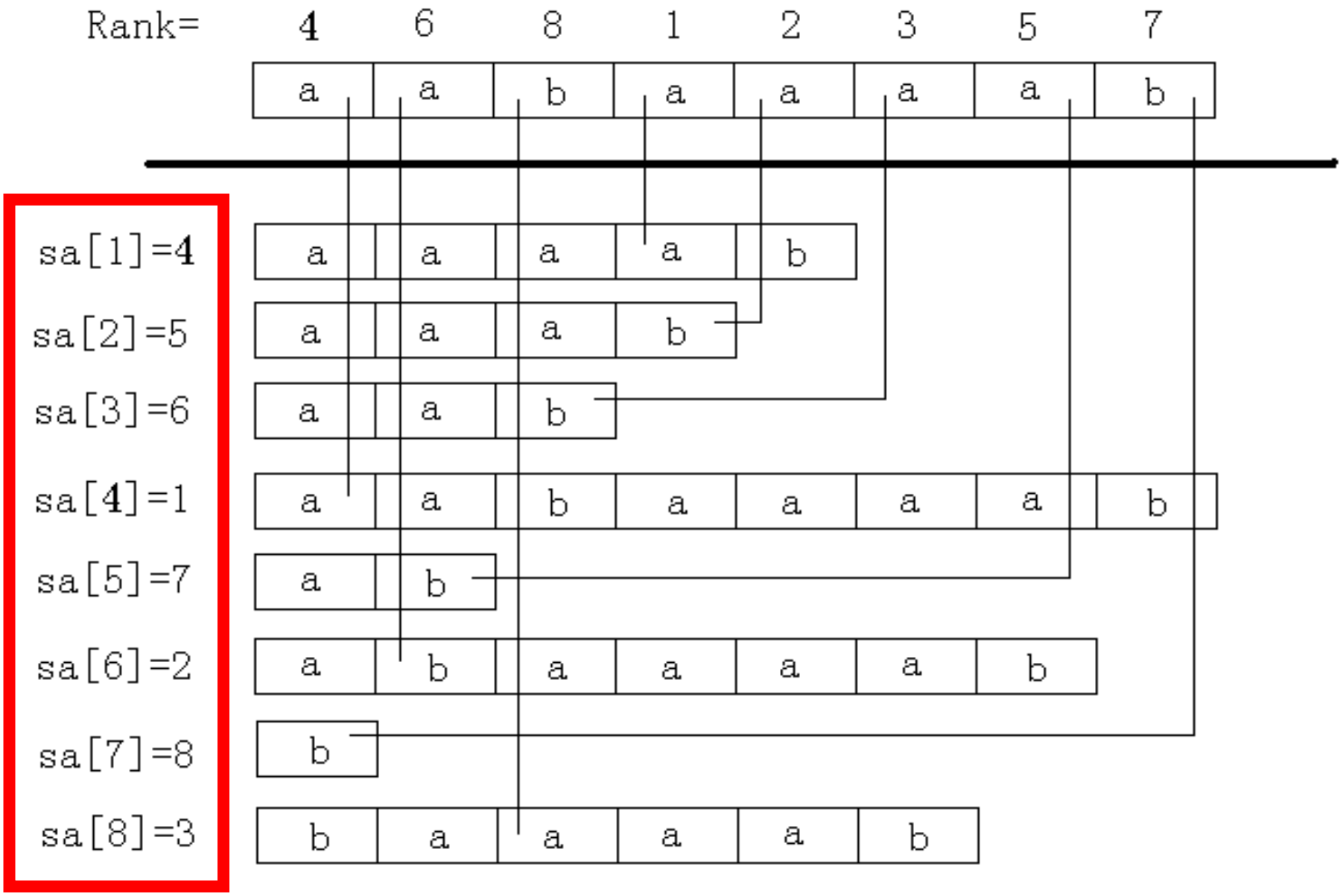

Rank= 4 6 8 1 2 3 5 7
a a b a a a a b
sa[1]=4
a a a a b
sa[2]=5
a a a b
sa[3]=6
a a b
sa[4]=1
a a b a a a a b
sa[5]=7
a b
sa[6]=2
a b a a a a b
sa[7]=8
b
sa[8]=3
b a a a a b

图1

DC3 算法

- ~~复杂，难以实现~~

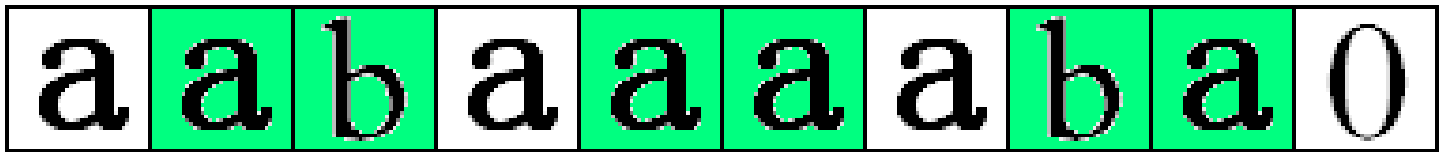
仅 40 行代码

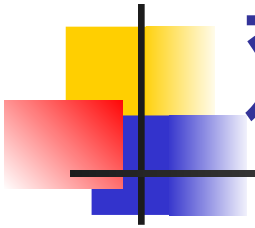
DC3 算法

- （１）、先将后缀分成两部分，然后对第一部分的后缀排序。
- （２）、利用（１）的结果，对第二部分的后缀排序。
- （３）、将（１）和（２）的结果合并，即完成对所有后缀排序。

（１）、将后缀分成两部分

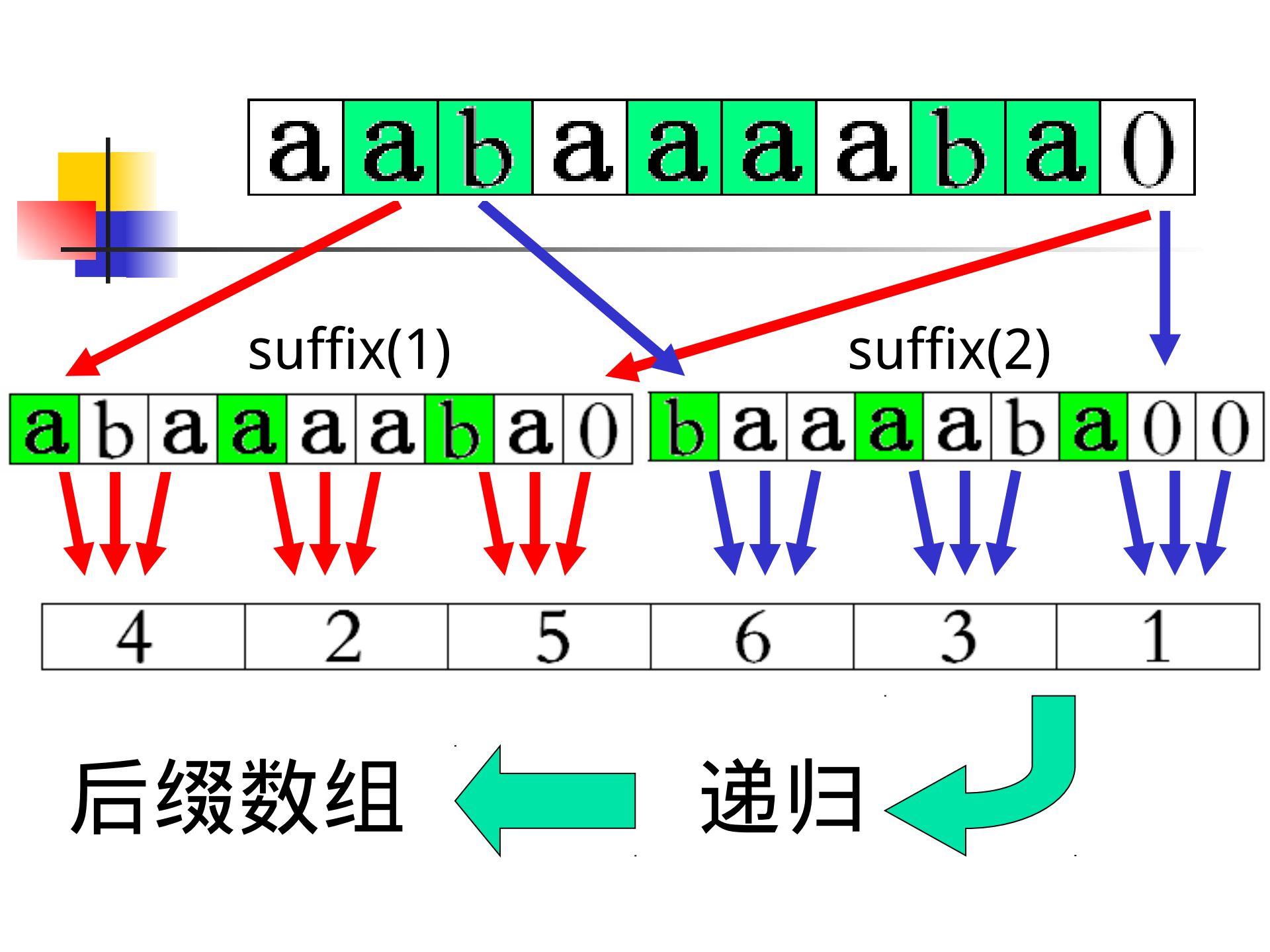
- 字符的编号从０开始。
- 将后缀分成两部分：
- 第一部分是后缀ｋ（ｋ模３不等于０）
- 第二部分是后缀ｋ（ｋ模３等于０）

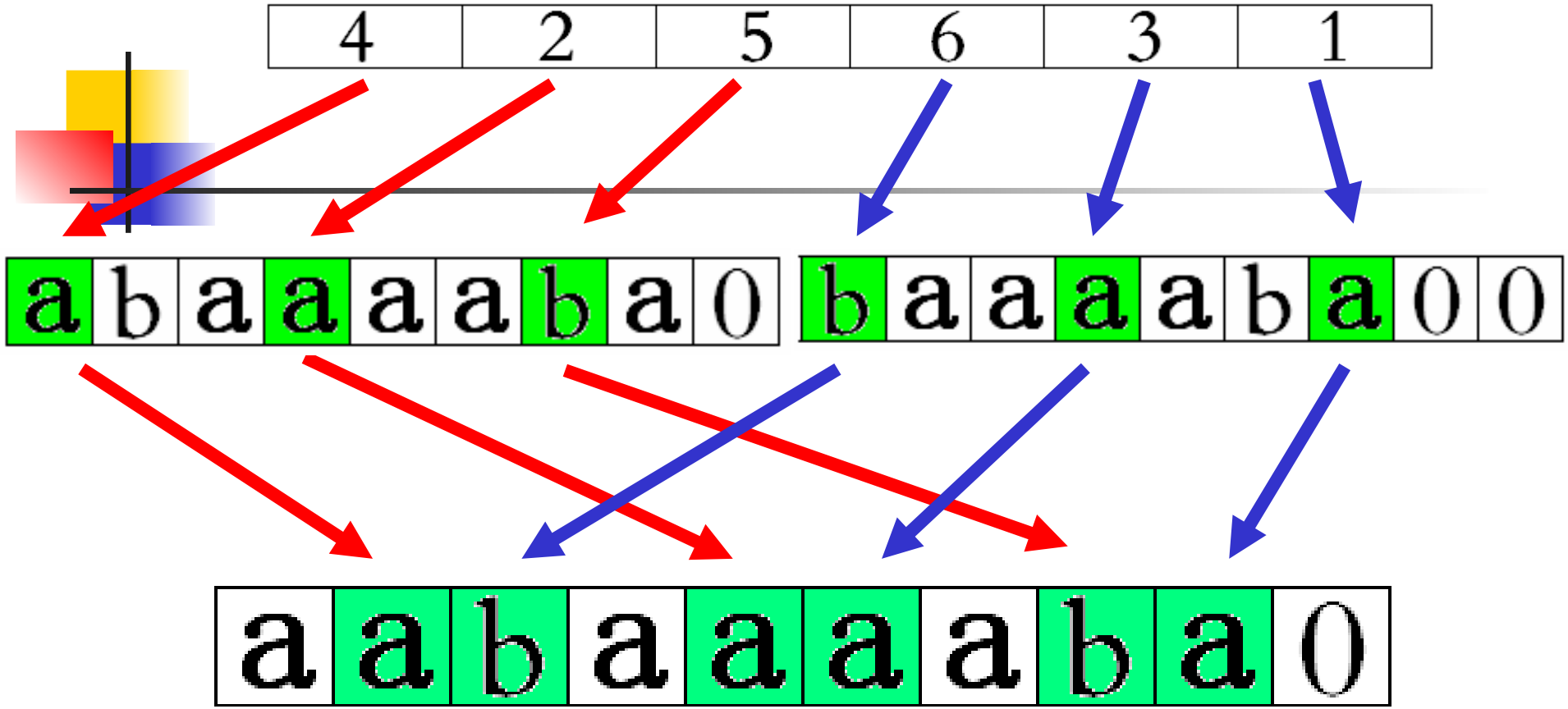




对第一部分的后缀排序。

- 为了方便接下来的操作，这里要求字符串必须以一个最小的字符结尾。


a a b a a a a b a 0
suffix(1)
suffix(2)
a b a a a a b a 0
b a a a a b a 0 0
4 2 5 6 3 1
后缀数组
递归

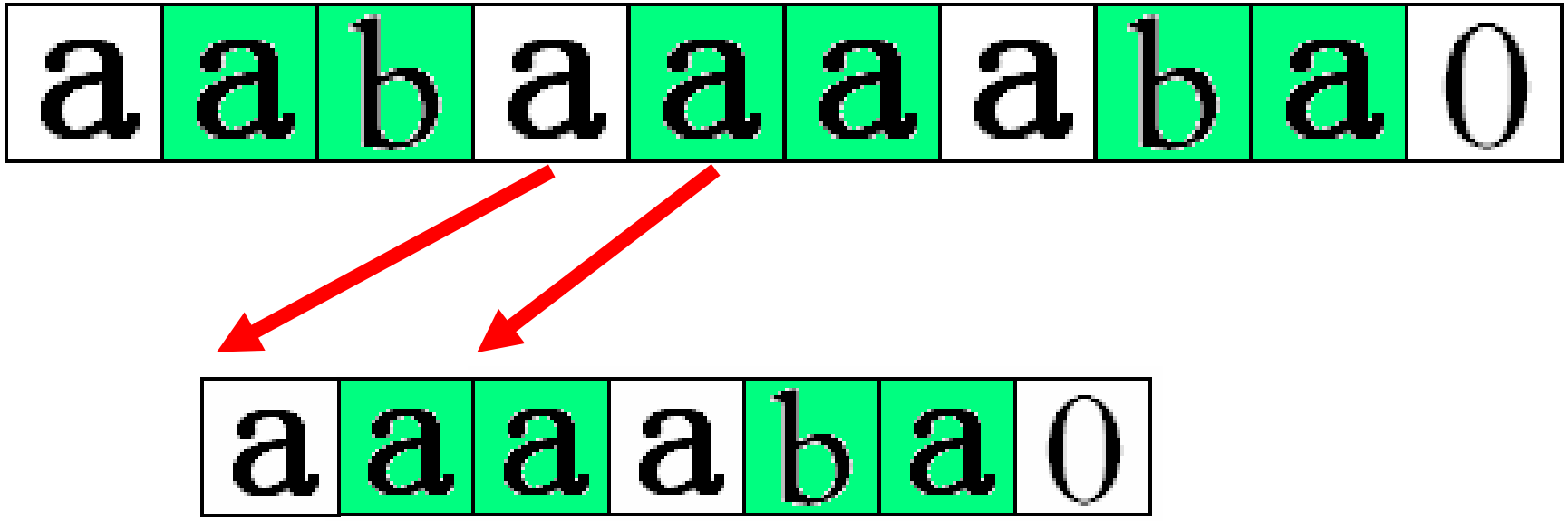


步骤（１）完成

DC3 算法

- （１）、先将后缀分成两部分，然后对第一部分的后缀排序。
- （２）、利用（１）的结果，对第二部分的后缀排序。
- （３）、将（１）和（２）的结果合并，即完成对所有后缀排序。

（２）、对第二部分的后缀排序



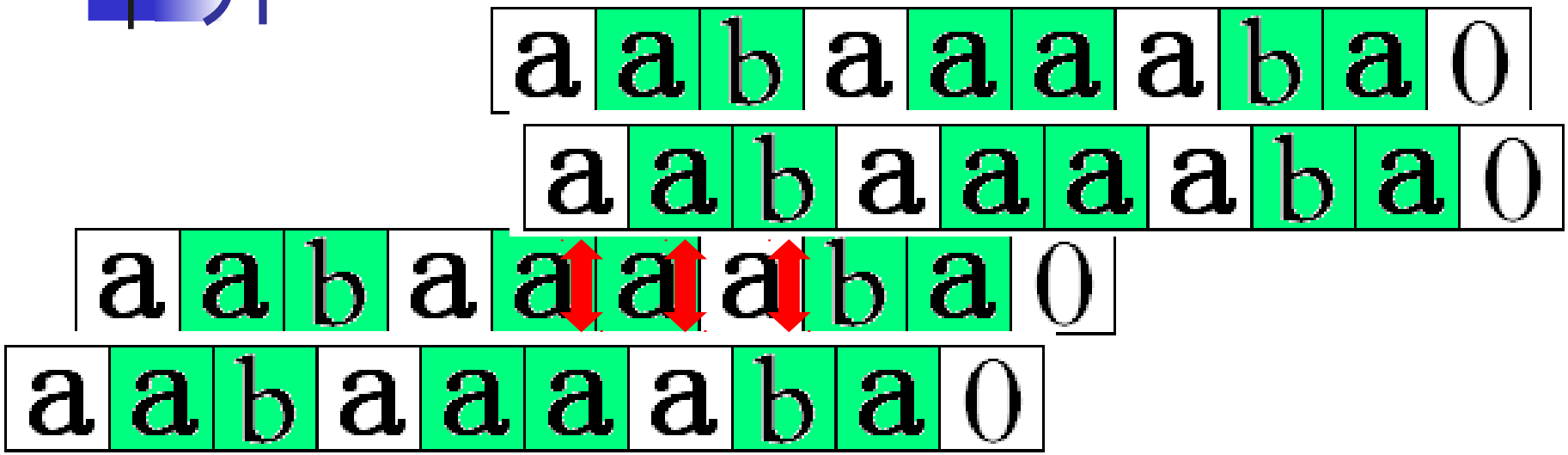
基数排序即可

步骤（２）完成

DC3 算法

- （１）、先将后缀分成两部分，然后对第一部分的后缀排序。
- （２）、利用（１）的结果，对第二部分的后缀排序。
- （３）、将（１）和（２）的结果合并，即完成对所有后缀排序。

（３）、将(1)和(2)的结果合并



每次的比较都可以高效的完成

步骤（３）完成

算法分析

- 假设这个算法的时间复杂度为 f(n)。
- 第（ 1 ）步排序的时间为 O(n)（一般来说， m 比较小，这里忽略不计），新的字符串的长度不超过 2n/3 ，求新字符串的 sa 的时间为 f(2n/3)。
- 第（ 2 ）和第（ 3 ）步的时间都是 O(n)。

算法分析

$$f(n) = O(n) + f(2n/3)$$

$$f(n) \le c \times n + f(2n/3)$$

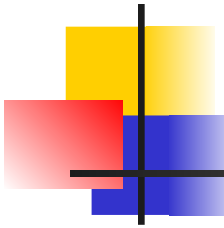
$$f(n) \le c \times n + c \times (2n/3) + c \times (4n/9) + \ldots\ldots \le 3c \times n$$

所以 $f(n)=O(n)$

- 由此看出，DC3 算法是一个优秀的线性算法！

后缀数组的应用

- 例 1 ：如果字符串 L 同时出现在字符串 A 和字符串 B 中，则称字符串 L 是字符串 A 和字符串 B 的公共子串。
- 给定两个字符串 A 和 B ，求最长公共子串。
- 例如：字符串“ aaaba” 和“ abaa” 的最长公共子串为“ aba”



height 数组

- 定义
- height[i]=suffix(sa[i-1]) 和 suffix(sa[i]) 的最长公共前缀，也就是排名相邻的两个后缀的最长公共前缀。
- 那么对于 j 和 k ，不妨设 rank[j]<rank[k], 则有以下性质：

height 数组

- suffix(j) 和 suffix(k) 的最长公共前缀为 height[rank[j]+1], height[rank[j]+2], height[rank[j]+3], … ,height[rank[k]] 中的最小值。
- 例如，字符串为“ aabaaaab” ，求后缀“ abaaaab” 和后缀“ aaab” 的最长公共前缀。

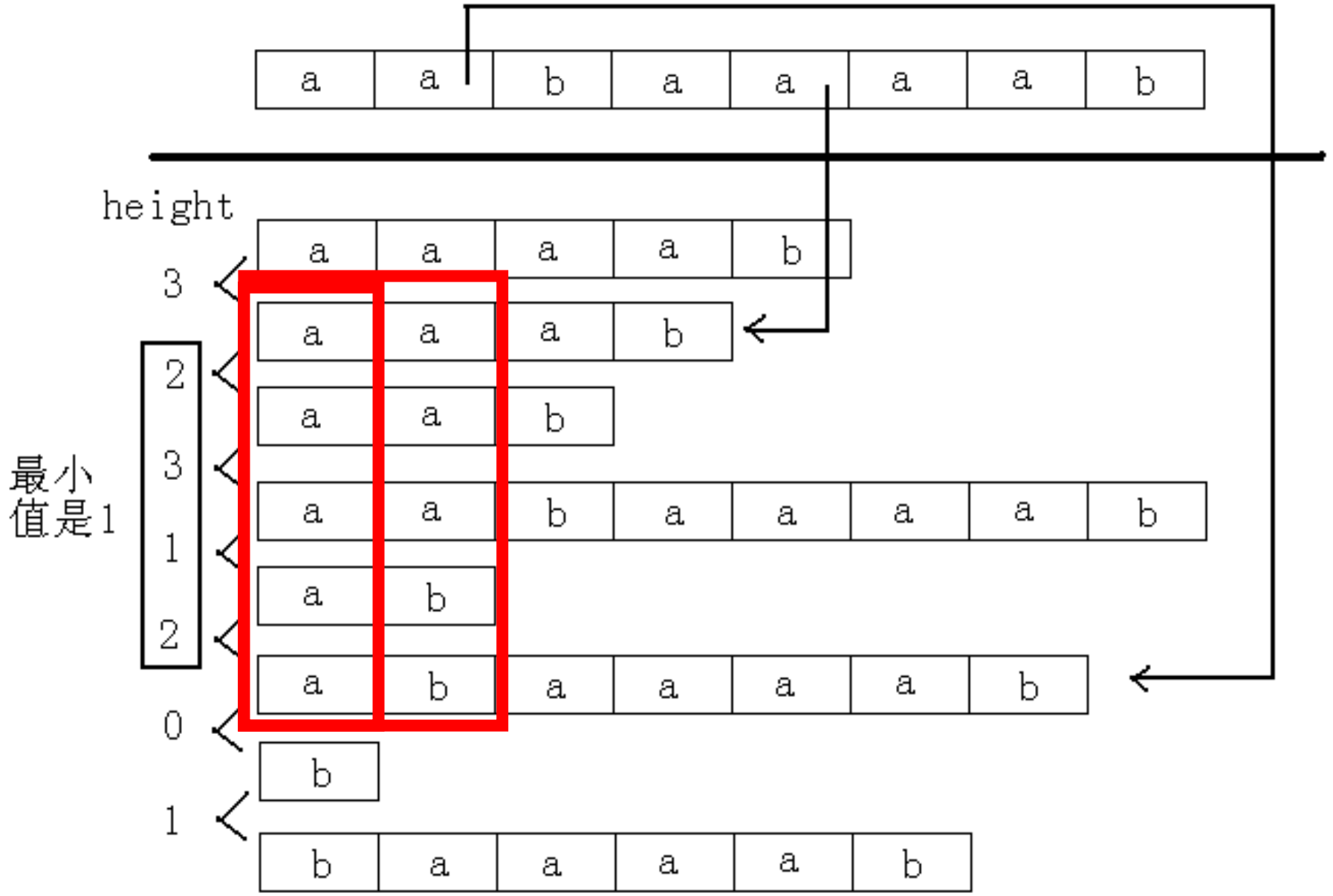

a a b a a a a b
height
3
2
3
1
2
0
1
最小
值是1
a a a a b
a a a b
a a b
a a b a a a a b
a b
a b a a a a b
b
b a a a a b

图4

例 1 最长公共子串

- 字符串的任何一个子串都是这个字符串的某个后缀的前缀。求 A 和 B 的最长公共子串等价于求 A 的后缀和 B 的后缀的最长公共前缀的最大值。
- 将第二个字符串写在第一个字符串后面，中间用一个没有出现过的字符隔开，再求这个新的字符串的后缀数组。

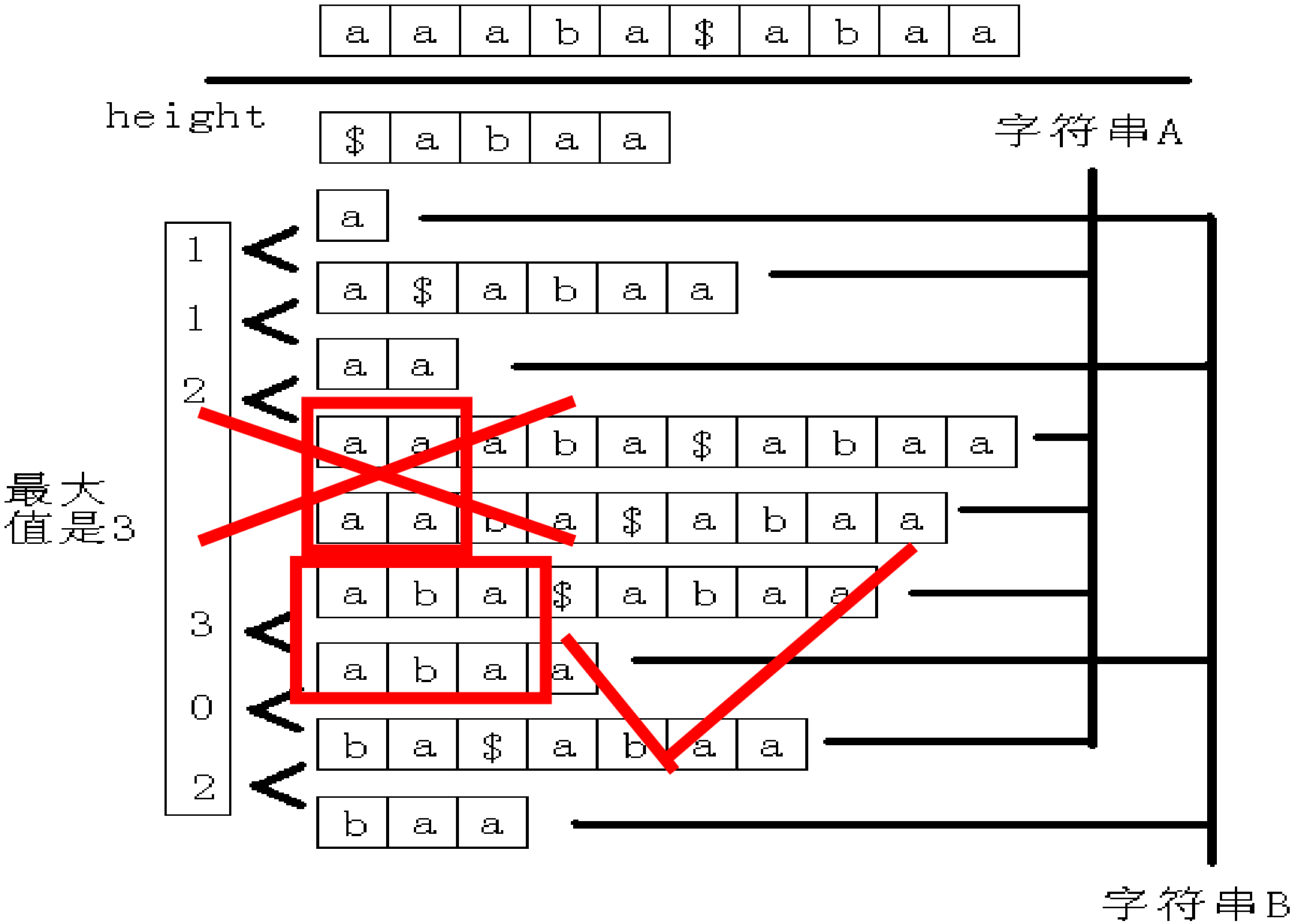

a a a b a $ a b a a
height
$ a b a a
字符串A
a
a $ a b a a
a a
a a a b a $ a b a a
a a b a $ a b a a
a b a $ a b a a
a b a a
b a $ a b a a
b a a
1
1
2
3
0
2
最大
值是3
字符串B

图8

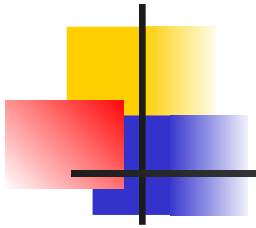
如何高效的计算 height 数组？

- 如果按 height[2] ， height[3] ，…… ， height[n] 的顺序计算，最坏情况下时间复杂度为 $O(n^2)$ 。这样做并没有利用字符串的性质。定义 h[i]=height[rank[i]] ，也就是 suffix(i) 和在它前一名的后缀的最长公共前缀。 h 数组有以下性质：

h[i] ≥ h[i-1]-1

最长公共前缀至少为 h[i-1]-1

如何高效的计算 height 数组？

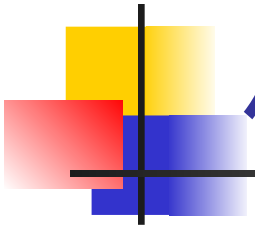


- 按照 h[1],h[2],……,h[n] 的顺序计算，并利用 h 数组的性质，时间复杂度可以降为 O(n)。

例 1 最长公共子串

- （ 1 ）连接两个字符串
- （ 2 ）求后缀数组
- （ 3 ）求 height 数组
- （ 4 ）求排名相邻但原来不在同一个字符串中的两个后缀的 height 值的最大值。
- 时间复杂度已经取到下限，由此看出，这是一个非常优秀的算法。

$O(|A|+|B|)$



总结

- 后缀数组是字符串处理中非常优秀的数据结构，是一种处理字符串问题的有力工具。
- 我们应该掌握好后缀数组这种数据结构，并且能在不同类型的题目中灵活、高效的运用。

更多内容……

- 第一部分 后缀数组的实现
- (1) 倍增算法　时间复杂度 O(nlogn)
- (2) DC3 算法　时间复杂度 O(n)
- 第二部分 后缀数组的应用
- (1) 最长公共前缀
- (2) 单个字符串的相关问题
- (3) 两个字符串的相关问题
- (4) 多个字符串的相关问题

完整的代码

25 行

40 行

共 13 个例题

原题

解答

参考程序