Sone2 解题报告

绍兴一中 王鉴浩

1 试题来源

原创

2 试题大意

给一个长 n 的 a 串和一个长 m 的 b 串。

定义一个关于 a 串和 b 串的函数 $F(a,b)$，$F(a,b)$ 的值是一个二元组 (x,y)，x,y 的含义如下：

- 设 $a[l:r]$ 表示 a 串第 l 个字符到第 r 个字符之间的子串。特别地，当 $l>r$ 时，$a[l:r]$ 表示空串。再定义一个关于 s 串和 t 串的函数：

$$f(s,t)=\max_{s[1:k]=t[1:k]} k$$

则

$$x=\max_{i=1}^{|a|} f(a[i:|a|],b)$$

- y 表示当 x 最大化时，合法的 i 的个数。

有 q 个操作，操作有四种：

1. 改变 a 串的某一位，询问 $F(a,b)$。
2. 选择 a 串中的一个子串 c，并询问 $F(c,b)$。
3. 选择 b 串的两个位置，并询问这两个后缀的最长公共前缀的长度。

4. 选择 b 串的两个子串 $s1, s2$，并询问串 $s1s2$ 是否是 b 串的子串，是的话输出 yes，否则输出 no。

3 数据范围

数据编号	n	m	q	字符集大小	备注
1,2	1000	≤ 100	1000	≤ 100000	
3,4	100000	≤ 100	100000		无操作二
5,6	100000	≤ 30	100000		
7,8	100000	≤ 100000	100000		只有操作三
9,10					只有操作四
11,12				5	b 串一定比例随机
13,14				5	b 串一定比例随机
15,16				5	b 串一定比例随机
17,18				≤ 100000	
19,20				≤ 100000	

4 算法介绍

此题一共有四个操作，本文会先按照操作三，操作四，操作一，操作二的顺序讲解。

4.1 操作三

假设我们询问的两个后缀在 b 串中的位置是 (x, y)。

我们可以把操作三定义成求

$$f\left(b\left[x:|b|\right], b\left[y:|b|\right]\right)$$

数据中有 10% 的数据只有操作三，运用算法二或算法三均可以通过这 10% 的数据。

4.1.1 算法一

我们可以直接枚举 f 函数中的 k，这个算法的时间复杂度为 $O(qm)$。

4.1.2 算法二

由于 b 串是不会变的。我们可以先用时间复杂度为 $O(m)$ 预处理 *hash* 数组。由于 f 函数中的 k 的取值是连续的。那么我们可以通过二分查找 k，用 *hash* 判断两个串是否相同。上述算法的时间复杂度为 $O(m + q\log m)$。

4.1.3 算法三

由于 b 串是不会变的。我们可以先用时间复杂度为 $O(m\log m)$ 预处理后缀数组和 *height* 数组。如果 $x = y$ 那么答案直接为 x 的后缀长度。否则我们可以通过后缀数组中的 *rank* 数组把区间 $[x, y]$ 映射到 *height* 数组上得到区间 $[l, r], l \le r$。那么我们可以根据 *height* 数组的定义把操作三重新定义成：

$$\min_{i=l+1}^{r} height[i]$$

那么问题就转化成 *rmq* 的问题了，我们可以通过用时间复杂度为 $O(m\log m)$ 预处理 ST 表，使得询问的时间复杂度为 $O(1)$。

这个算法的时间复杂度为 $O(m\log m + q)$。

4.2 操作四

假设询问的两个子串 $s1, s2$ 在 b 串中的区间是 $[l1, r1]$ 和 $[l2, r2]$。我们可以先把这两个子串拼起来成为串 s。然后我们要验证 $F(b, s)$ 的 x 是否为 $|s|$。一个加强操作：我们可以做操作四的时候，如果有解，我们可以同时计算出一个合法的位置。数据中有 10% 的数据只有操作四，运用算法三可以通过这 10% 的数据。

4.2.1 算法一

首先可以直接枚举 $F(b, s)$ 中的 i，然后验证 $f(b[i : |b|], s)$ 的值是否为 $|s|$。如果直接枚举 f 函数中的 k，那么这个算法的时间复杂度为 $O(qm^2)$。

4.2.2 算法二

可以发现算法一中的计算 f 函数的值本质就是操作三。所以如果运用操作三中的算法三，那么这个算法的时间复杂度为 $O(qm)$。

我们也可以把串 s 做 *kmp*，用串 b 匹配的时候计算 *newnext* 数组来统计答案。这个这个算法的时间复杂度也是 $O(qm)$。

4.2.3 算法三

我们考虑用后缀数组来组这个操作。

可以先用倍增算法在 $O(m\log m)$ 的时间复杂度内把串 b 做后缀数组，得到数组 *sa*，*rank* 和 *height*。然后我们发现，如果串 s 是 b 串的子串的话，那么串 s 就可以表示为 b 串中某一个后缀的前缀。于是，我们先把所有以 $s1$ 为前缀的后缀都找出来。根据 *sa* 数组的定义，可以发现这些后缀对应到 *sa* 数组上是连续的一段，这一段可以通过二分得到。现在要在这个区间中找到一个以 s 串为前缀的后缀。由于 *sa* 数组是把那些后缀按照字典序排序，那么又可以根据字典序，在这个区间中继续二分，寻找以 s 串为前缀的后缀。在二分的过程中，我们可以结合 *rank* 数组和 *LCP* 来进行快速判断。由于在操作三中，我们已经可以支持在 $O(1)$ 的时间复杂度内进行 *LCP* 的计算，那么二分的时间复杂度还是 $O(\log m)$。

上述解法的时间复杂度为 $O(m\log m + q\log m)$。

4.3 操作一

操作一为：把 a 串中的第 x 位改成 y。

4.3.1 算法一

我们把 b 串做 *kmp*，计算出 *next* 数组。对于每次询问，我们可以把 a 串直接在 b 串的 *next* 数组里匹配，计算出 *newnext* 数组，并统计 $F(a,b)$ 的答案。由于 *kmp* 的时间复杂度是线性的，那么上述算法的时间复杂度是 $O(q(n+m))$。

4.3.2 算法二

根据算法一，我们把它优化一下。我们发现有 30% 的数据 m 较小。而且我们可以发现，每次改 a 的第 x 位，只会影响 $x\sim min(n,x+m-1)$ 的 *newnext* 的值。那么我们只需要对这些位进行 *newnext* 重构就可以了。那么这个算法的时间复杂度为 $O(qm)$。

4.3.3 算法三

在这个算法中，我们需要先提出覆盖这个概念。

设模式串为长度为 m 的 b 串，文本串为长度为 n 的 a 串。

覆盖的基本思想为：我们用模式串 b 中的子串去精确覆盖 a 串。

下面给出覆盖的定义：

定义 4.1.

一个关于 b 串和 a 串的覆盖 $\mathcal{G}$，是一个字符串的序列 $[v_1, v_2, v_3...v_t]$，而 v_i 被称为是覆盖的一个元素。

覆盖需满足 $v_1v_2v_3...v_t = a$，且对于每个元素 v_i $(1 \le i \le t)$，均满足以下两条性质：

- 子串性质：v_i 是 b 串的子串。
- 极大化性质：若 $i < t$，则 v_iv_{i+1} 不是 b 串的子串。

特别地，对于 a 串的第 x 位，若 a_x 这个字符没有出现在 b 串中，我们也把 a_x 作为一个 v_i，不过这个 v_i 在 b 对应的是空串。

由于覆盖方式可能有多种，所以合法的 $\mathcal{G}$ 也可能有多种，具体实现时，我们只需取任意一种合法的 $\mathcal{G}$ 即可。

对于 $\mathcal{G}$ 我们可以简单地理解为 v_i 把 a 串分裂成了 t 个小块。

现在我们对于每个 v_i 用一个三元组 (s_i, l_i, r_i) 来表示：

- l_i, r_i 表示 v_i 在 b 串中的位置：$v_i = b[l_i : r_i]$。

  特殊地，若 v_i 在 b 串中表示的是空串，那么令 $l_i = r_i = 0$。

- s_i 表示 v_i 在 a 串中的位置，令 $s_i = 1 + \sum_{j=1}^{i-1} |v_j|$。

由于篇幅有限，关于 $\mathcal{G}$ 的初始化、动态维护和时间复杂度分析请参见我的集训队论文。

我们可以在 $O((n + m + q)\log m + q \log n)$ 的时间复杂度内维护 $\mathcal{G}$。

由于 $F(a, b)$ 函数的本质是把 $LCP(a[i : |a|], b)$ $(1 \le i \le |a|)$ 的最大值及最大值的个数表示成一个二元组，$\mathcal{G}$ 函数也可以简单地理解为把 a 串分裂

成了 t 个小块 v_j $(1 \le j \le t)$。对于每个 v_j，我们令 $s_j = v_j v_{j+1} v_{j+2} .. v_t$，并把 $LCP\left(s_j\left[i : |s_j|\right], b\right)$ $\left(1 \le i \le |v_j|\right)$ 的最大值及最大值的个数也表示成一个二元组w_j。于是我们就可以通过合并部分的最大值及最大值的个数 w_j，来得到整体的最大值及最大值的个数 $F(a, b)$。

而且根据 v_j 的极大化性质，我们可以直观地发现当 $s_j = v_j v_{j+1} v_{j+2}$ 时，w_j 的值并不会变。于是 w_j 只和 v_j, v_{j+1}, v_{j+2} 有关。在上一节中，我们已经可以用 $O(\log n)$ 的时间复杂度找到 v_j 的前驱和后继，那么，每当我们需要计算 w_j 时，只需要耗费 $O(\log n)$ 的时间就能找到 v_{j+1} 和 v_{j+2}。

小结一下，我们把计算 $F(a, b)$ 的问题简化成了计算 w_j。而对于计算 w_j，我们又可以把问题转化为：给出 b 串的 3 个子串 s_1, s_2, s_3，令 $s = s_1 s_2 s_3$，把 $LCP\left(s\left[i : |s|\right], b\right)$ $(1 \le i \le |s_1|)$ 的最大值及最大值的个数合并。

这个问题是一个关于 *Border Tree* 的经典问题。由于篇幅有限，这个问题的解法请读者参见我的集训队论文。我们可以在 $O((n + m + q) \log m + q \log n)$ 的时间复杂度内来对这个新问题进行计算。

4.4 操作二

假设询问的 a 串中的区间为 $[x, y]$。

4.4.1 算法一

我们可以效仿操作一中的算法一。在 $[x, y]$ 这个区间中计算 *newnext* 数组，并统计答案。那么这个算法的时间复杂度是 $O(qn)$。

4.4.2 算法二

如果我们运用操作一中的算法二。我们用线段树来维护这个 *newnext* 数组。每次操作对于区间左端 m 位的 *newnext* 重新计算，剩下运用线段树统计答案，然后把答案合并。那么操作二的时间复杂度为 $O(qm + q \log n)$。操作一的时间复杂度为 $O(qm \log n)$。

4.4.3 算法三

对于第二种操作，一个简单的想法就是我们维护 $a[L : R]$ 这个串的覆盖 $\mathcal{G}'$，

然后进行计算。由于 $a[L:R]$ 是 a 串的子串，我们可以发现，如果把 $\mathcal{G}'$ 序列的头尾两个元素去掉，那么这个序列就是 $\mathcal{G}$ 序列的子序列了。于是对于第二种操作，只需要对于 $\mathcal{G}'$ 的头尾两个元素进行特殊计算，而中间的部分我们可以运用数据结构直接从 $\mathcal{G}$ 序列中提取。那么我们就可以在单次 $O(\log n + \log m)$ 的时间复杂度内进行构造 $\mathcal{G}'$ 了。然后在维护 v'_j 的同时维护 w'_j，我们就可以直接对 $\mathcal{G}'$ 进行计算了。

于是，此题就可以在 $O((n+m+q)\log m + q\log n)$ 的时间复杂度内解决了。

5 算法总结

对于每一类数据，我都列出了每种操作的所需最低算法类型，见下表：

数据编号	操作一	操作二	操作三	操作四
1,2	一	一	一	二
3,4	二	二	一	二
5,6	二	二	一	二
7,8	无	无	二	无
9,10	无	无	无	三
11,12	三	三	二	三
13,14				
15,16				
17,18				
19,20				

本文提出的解法可以在时间复杂度在 $O((n+m+q)\log m + q\log n)$ 内解决此题。对于一个静态模版串和动态文本串的匹配这一类问题都是可以先把文本串变成模版串的一个覆盖 $\mathcal{G}$，然后通过一些数据结构来维护 $\mathcal{G}$，使得把问题转化为一个只关于模版串的新问题。对于这个新问题，我们可以使用 *Border Tree* 来进行在线高效操作，一次匹配的时间复杂度为 $O(\log m)$。而当模版串随机的话，由于此时如果把模版串建 *KMP* 自动机，自动机中的树高是 $O(\log m)$ 级别的，所以可以使用 *KMP* 自动机来代替 *BorderTree* 进行计算答案。那么就能大大降低实现这类问题的难度。

在集训队互测中，预计会有 4 人能得到 50 分，有 6 人能得到 40 分，有 2 人能得到 20 分。