小C的后缀数组解题报告

绍兴市第一中学 洪华敦

摘 要

2016年集训队互测第三场中，一道传统字符串题的命题思路和题目分析

1 题目大意

小C写了一个后缀数组求两个后缀最长公共前缀(lcp)的程序，他的算法流程是这样子的：

1.读入一个长度为n的字符串$a[1..n]$

2.定义数组$sa[1..n]$，$sa[i]$表示将所有后缀排序后，第i小的后缀是$a[sa[i]..n]$，定义两个字符串的比较方式：逐位比较，如果一个字符串是另一个字符串的前缀，那么前者较小

3.定义数组$rank[1..n]$，令$rank[sa[i]] = i$

4.对于给定的询问l, r，设$x = min(rank[l], rank[r])$，设$y = max(rank[l], rank[r])$

5.对于每个满足$x \le i < y$的i，求出$lcp(a[sa[i]..n], a[sa[i+1]..n])$，最小值就是答案

（注意，第5步中的lcp，使用的是一个暴力的算法，与传统后缀数组的求$heigh$方法无关，你不需要关心他如何实现）

原本这是一个相当完美的算法，但是有个熊孩子在第二步和第三步中间加了一步：将sa数组随机打乱，即sa数组变成了一个随机生成的排列

现在给定字符串和询问，小C想知道被熊孩子打乱后的期望输出是什么

为了避免精度误差，你需要将答案乘上$n!$后对998244353取模后输出

2 读入格式

第一行两个整数n，Q表示串长和询问个数。

第二行一个长度为n的小写字母串。

接下来Q行，每行两个正整数l,r，保证$l \neq r$。

3 输出格式

输出Q行，每行一个整数表示答案。

4 数据范围

对于10%的数据，有$1 \leq n \leq 10$，$1 \leq Q \leq 10$。

对于15%的数据，有$1 \leq n \leq 100$，$1 \leq Q \leq 2$。

对于15%的数据，有$1 \leq n \leq 100$，$1 \leq Q \leq 100$。

对于10%的数据，有$1 \leq n \leq 3 * 10^3$，$1 \leq Q \leq 3 * 10^3$。

对于20%的数据，有$1 \leq n \leq 10^5$，$Q = 1$。

对于30%的数据，有$1 \leq n \leq 10^5$，$1 \leq Q \leq 10^5$。

以上数据组两两互不相交。

5 总体分析

5.1 得分估计

对于集训队员而言，所有队员都应该得到至少10%的分数

预计会有9人得到至少50%的分数，至少5人得到100%的分数

5.2 考点分析

本题考察了选手对后缀数据结构和快速傅里叶变换的掌握程度

6 算法介绍

6.1 算法1

题目中提到，顺序数组是随机打乱的，我们可以$O(n!)$枚举顺序数组，然后

找到l, r的位置，并两两计算最长公共前缀，时间复杂度$O(Qn!n^2)$。

可以注意到，我们在计算最长公共前缀上花了很多时间，我们可以预处理，对n^2组后缀预处理出最长公共前缀，于是复杂度可以优化到$O(Qn!n)$。

6.2 算法2

对于本题，我们有一个推论：设一个后缀$s[i : n]$的字典序大小排名是$rank[i]$，并且令$sa[rank[i]] = i$。

则设顺序数组中，$[l..r]$内字典序最大的后缀是y，最小的是x，则两两最长公共前缀的最小值是$s[x : n]$和$s[y : n]$的最长公共前缀。

下面我们来证明这个推论：

设$lcp(x, y)$表示$s[x : n]$和$s[y : n]$的最长公共前缀，设$h[i] = lcp(sa[i], sa[i + 1])$。

根据经典的后缀数组算法，我们有：

$$lcp(x, y) = lcp(y, x) \quad (rank[x] > rank[y])$$

$$lcp(x, y) = min(h[rank[x]], h[rank[x] + 1]...h[rank[y] - 1]) \quad (rank[x] < rank[y])$$

设顺序序列中的l到r中间的后缀按顺序分别是$a_1..a_m$。

对于每个满足$rank[x] \le i < rank[y]$的i，只需要证明存在一个j，使得$h[i]$对$lcp(a_j, a_{j+1})$产生过贡献即可。

我们先找到$a_j = y$，那么设a_{j-1}和a_{j+1}中$rank$较小的一个是p，则$h[rank[p]]..h[rank[y]-1]$都出现过了，于是我们可以去掉y，归纳下去，最后所有满足上面条件的$h[i]$一定都出现过。

于是我们只要枚举最小后缀x，最大后缀y，然后计算随机序列中，l,r中字典序极值是x, y的方案数。

我们这里只讨论l, r, x, y互不相同的情况，有相同的情况类似。

方案数就是

$$\sum_{k=0}^{rank[y]-rank[x]-3} C_{rank[y]-rank[x]-3}^{k} * (k + 2)! * (n - k - 3)! * 2$$

于是我们预处理组合数和阶乘，枚举了x, y后暴力计算即可。

时间复杂度$O(Qn^3 + n^2)$。

6.3 算法3

可以发现，式子$\sum_{k=0}^{rank[y]-rank[x]-3} C_{rank[y]-rank[x]-3}^{k} * (k+2)! * (n-k-3)! * 2$的值只跟$rank[y] - rank[x] - 3$有关。

令

$$f[x] = \sum_{k=0}^{x} C_x^k * (k+2)! * (n-k-3)! * 2$$

我们可以用整数域下的快速傅里叶变换在$O(nlogn)$的时间内计算出$f[x]$。

具体步骤如下，令

$$A[x] = \frac{(k+2)!(n-k-3)!}{k!}$$

$$B[x] = \frac{2}{x!}$$

对A, B做卷积可以得到$c[x] = \frac{f[x]}{x!}$。

于是时间复杂度降到了$O(Qn^2 + nlogn)$。

6.4 算法4

之前的算法3中，我们计算答案的方法是累计$lcp(x, y) * f[rank[y] - rank[x] - 3]$，这样复杂度是$O(Qn^2)$的，我们可以发现，根据后缀数组的性质，不同的$lcp(x, y)$最多只有$O(n)$种。

那么现在我们不枚举x, y，改为枚举$lcp(x, y)$，具体就是枚举$lcp(x, y) = h[i]$，我们找出pl, pr，满足$h[i]$是$h[pl..pr]$中的最小值。

这个区间有贡献，当且仅当：

$$rank[x] \in [pl, pr+1]$$

$$rank[y] \in [pl, pr+1]$$

$$min(rank[x], rank[y]) \le i < max(rank[x], rank[y])$$

那我们的最小值的取值范围就是$[pl, min(rank[l], rank[r]))$，最大值的取值范围就是$(max(rank[l], rank[r]), pr + 1]$了。

相当于求

$$\sum_{i=1}^{A}\sum_{j=1}^{B} f[i + j + D]$$

其中D是一个常数。

我们设

$$f^2[x] = \sum_{i=1}^{x} f[i]$$

$$f^3[x] = \sum_{i=1}^{x} f^2[i]$$

$$\sum_{i=1}^{A}\sum_{j=1}^{B} f[i+j+D] = \sum_{i=1}^{A} f^2[i+B+D]-f^2[i+D] = f^3[A+B+D]-f^3[B+D]-f^3[A+D]+f^3[D]$$

于是我们可以通过预处理，$O(1)$计算出方案数。

时间复杂度$O(Qn)$。

6.5 算法5

之前我们所有算法都是建立在后缀数组这个算法的基础上的，众所周知，后缀数组是后缀树的简化版，我们可以在后缀树上考虑这个问题。

首先我们可以用经典的后缀自动机算法构造出后缀树。

设顺序序列中$l..r$中的后缀在后缀树上的结点是$a_1, a_2..a_m$，在后缀树上，两个串的最长公共前缀就是对应结点的最近公共祖先的深度。

为了方便表达，下面$LCA(l, r)$表示的是后缀l,r代表的结点的LCA。

而$LCA(a_i, a_{i+1})$中最浅的结点显然就是$LCA(a_1...a_m)$。

那么由于a中必定有后缀l, r对应的结点，我们可以枚举$LCA(l, r)$的祖先，然后计算贡献。

如何计算贡献呢，对于一个祖先x，设他的子树中有$size[x]$个后缀结点，设他的儿子y也是$LCA(l,r)$的祖先。

首先l,r对应的结点是必选的，但是为了使得x成为LCA，我们必须再多选一些点，这里我们可以用容斥思想，在子树x中随便选其他点，对于一个不合法的方案，他选的所有点必然在子树y中，减掉即可。

即方案数就是：

$$2*\sum_{k=0}^{size[x]-2} C_{size[x]-2}^{k} k!(n-k-1)! - 2*\sum_{k=0}^{size[y]-2} C_{size[y]-2}^{k} k!(n-k-1)!$$

设$g[x] = 2*\sum_{k=0}^{x-2} C_{x-2}^{k} k!(n-k-1)!$，$g$我们可以用快速傅里叶变换在$O(nlogn)$内求出。

于是预处理好数组g，然后枚举每个祖先，根据子树里的后缀结点数量，直接计算即可。

时间复杂度$O(Qn + nlogn)$。

6.6 算法6

以上算法的瓶颈是每次要枚举所有祖先，对于每个结点去计算答案，在树退化成链时复杂度会很劣

如果我们可以做到一个结点的贡献只和他本身有关且独立，那我们只要链求和就行了

我们可以发现，一个贡献的组成是两部分，一个是父亲结点，一个是儿子结点。这个贡献可以映射到连接他们的那条边上

于是我么计算答案只需要求出LCA后求LCA到根的所有边的贡献之和即可

我们可以用$O(nlogn)$的时间预处理，之后询问时用倍增的方法求出LCA

于是问题就完美地解决了。

时间复杂度$O(Qlogn + nlogn)$