一道有趣的字符串题

《基因组重构》问题拓展

laofu wh zzt wxh010910

陈江伦(长郡中学) 朱震霆(安徽师范大学附属中学) 王修涵(成都七中)

February 7, 2018

一句话题意

给一个长度为n，字符集也为n的字符串S，m次询问$S[l...r]$的后缀树节点数。
$n \leq 10^5$，$m \leq 3 \times 10^5$。

前置技能

SAM，LCT，线段树

SAM生成树，启发式合并，二分+哈希，倍增

扫描线，树套树/CDQ分治+线段树

前置技能

SAM，LCT，线段树

SAM生成树，启发式合并，二分+哈希，倍增

扫描线，树套树/CDQ分治+线段树
~~上面内容很简单相信大家都会~~

后缀自动机

后缀自动机是一个能接收所有后缀的自动机，它的一个节点表示一堆右端点集合相同的串，比如$abaab$中b和ab的右端点集合都为$\{2,5\}$，所以它们会被压缩到一个节点。

后缀自动机

后缀自动机是一个能接收所有后缀的自动机，它的一个节点表示一堆右端点集合相同的串，比如$abaab$中b和ab的右端点集合都为$\{2,5\}$，所以它们会被压缩到一个节点。
事实上一个节点能表示的串的长度是一个区间，而如果右端点集合$A \in B$，那么表示B的节点是表示A的节点的祖先，按照这个方法可以构造出一棵$parent$树。

后缀自动机

后缀自动机是一个能接收所有后缀的自动机，它的一个节点表示一堆右端点集合相同的串，比如$abaab$中b和ab的右端点集合都为$\{2,5\}$，所以它们会被压缩到一个节点。
事实上一个节点能表示的串的长度是一个区间，而如果右端点集合$A \in B$，那么表示B的节点是表示A的节点的祖先，按照这个方法可以构造出一棵$parent$树。
可以证明后缀自动机节点数和转移数是$O(n)$的，可以在$O(nlog\sum)$的时间内完成构造。

后缀自动机

后缀自动机是一个能接收所有后缀的自动机，它的一个节点表示一堆右端点集合相同的串，比如$abaab$中b和ab的右端点集合都为$\{2,5\}$，所以它们会被压缩到一个节点。
事实上一个节点能表示的串的长度是一个区间，而如果右端点集合$A \in B$，那么表示B的节点是表示A的节点的祖先，按照这个方法可以构造出一棵$parent$树。
可以证明后缀自动机节点数和转移数是$O(n)$的，可以在$O(nlog\sum)$的时间内完成构造。
如果想要系统学习后缀自动机，可以参考陈老师课件。

记号定义

对于一个字符串S，记$S[i]$表示S的第i个字符，$S[l,r]$表示S位于区间$[l,r]$的子串，$|S|$表示S的长度。

记号定义

对于一个字符串S，记$S[i]$表示S的第i个字符，$S[l,r]$表示S位于区间$[l,r]$的子串，$|S|$表示S的长度。
在后缀自动机中，记mx_i表示i能表示的串的最大串长，mi_i表示i能表示的串的最小串长，fa_i表示i在$parent$树上的父亲，那么$mi_i = mx_{fa_i} + 1$。$right_i$表示i表示的串的右端点集合，$last_i$表示$\max right_i$。

记号定义

对于一个字符串S，记$S[i]$表示S的第i个字符，$S[l,r]$表示S位于区间$[l,r]$的子串，$|S|$表示S的长度。
在后缀自动机中，记mx_i表示i能表示的串的最大串长，mi_i表示i能表示的串的最小串长，fa_i表示i在$parent$树上的父亲，那么$mi_i = mx_{fa_i} + 1$。$right_i$表示i表示的串的右端点集合，$last_i$表示$\max right_i$。
在用增量法构造后缀自动机的过程中，假设添加一个字符c，令p表示上一次添加字符的叶节点所在$parent$链中具有字符c出边的深度最深的结点，用np结点表示加入当前字符后建立的叶子节点，用q表示p沿字符c出边到达的点，nq表示当不满足$mx_p + 1 = mx_q$时新建的结点。

部分分做法

emmm...相信大家都知道后缀树节点数等于反串的后缀自动机节点数。
为了方便，之后默认将已经reverse过S了。

部分分做法

emmm...相信大家都知道后缀树节点数等于反串的后缀自动机节点数。
为了方便，之后默认将已经reverse过S了。
所以直接暴力就行了，复杂度$O(nm)$。

部分分做法

emmm...相信大家都知道后缀树节点数等于反串的后缀自动机节点数。
为了方便，之后默认将已经reverse过S了。
所以直接暴力就行了，复杂度$O(nm)$。
以上内容相信大家都会。

性质分析

后缀自动机的节点分为np和nq两类，那么不难想到对两类分别计数，np节点的个数就是字符串长度，表示S的每一个前缀。

性质分析

后缀自动机的节点分为np和nq两类，那么不难想到对两类分别计数，np节点的个数就是字符串长度，表示S的每一个前缀。那么nq节点的出现原因是什么呢？后缀自动机的本质是将右端点相同，且出现次数相同的子串进行合并，而nq节点的出现，就是为了区分右端点相同，但出现次数不同的子串。

性质分析

后缀自动机的节点分为np和nq两类，那么不难想到对两类分别计数，np节点的个数就是字符串长度，表示S的每一个前缀。
那么nq节点的出现原因是什么呢？后缀自动机的本质是将右端点相同，且出现次数相同的子串进行合并，而nq节点的出现，就是为了区分右端点相同，但出现次数不同的子串。
这给我们提供了一个思路：对于两个串A，B，如果A是B的后缀并且$|A|+1=|B|$，且存在字符$c\neq B[1]$，满足cA是S的子串，那么我们就需要给A单独建立一个nq节点来区分。

扫描线

考虑一个经典思路：扫描线，枚举右端点r，维护每个左端点l的答案。

扫描线

考虑一个经典思路：扫描线，枚举右端点r，维护每个左端点l的答案。

按照刚刚的性质，我们可以打一个暴力：枚举所有新生成的子串$S[k,r]$，假设其长度为$m=r-k+1$，首先判断$S[k,r]$最后一次出现的位置$last$，如果$last$不存在或者$S[r-m]=S[last-m]$，那么不需要新建nq。

扫描线

考虑一个经典思路：扫描线，枚举右端点r，维护每个左端点l的答案。

按照刚刚的性质，我们可以打一个暴力：枚举所有新生成的子串$S[k,r]$，假设其长度为$m=r-k+1$，首先判断$S[k,r]$最后一次出现的位置$last$，如果$last$不存在或者$S[r-m]=S[last-m]$，那么不需要新建nq。

否则，找到最大的t满足$S[t-m+1,t]=S[k,r]$且$S[t-m]\neq S[r-m]$，然后找到最大的p满足$S[p-m+1,p]=S[k,r]$且$S[p-m]\neq S[r-m], S[p-m]\neq S[t-m]$，如下图：

扫描线

考虑一个经典思路：扫描线，枚举右端点r，维护每个左端点l的答案。

按照刚刚的性质，我们可以打一个暴力：枚举所有新生成的子串$S[k,r]$，假设其长度为$m=r-k+1$，首先判断$S[k,r]$最后一次出现的位置$last$，如果$last$不存在或者$S[r-m]=S[last-m]$，那么不需要新建nq。

否则，找到最大的t满足$S[t-m+1,t]=S[k,r]$且$S[t-m]\neq S[r-m]$，然后找到最大的p满足$S[p-m+1,p]=S[k,r]$且$S[p-m]\neq S[r-m], S[p-m]\neq S[t-m]$，如下图：

如果$l\leq p-m$，我们在$r=t$时已经为它新建过nq节点了，而对于$p-m<l\leq t-m$，我们需要对它新建一个nq节点，即区间加一。

扫描线

考虑一个经典思路：扫描线，枚举右端点r，维护每个左端点l的答案。

按照刚刚的性质，我们可以打一个暴力：枚举所有新生成的子串$S[k,r]$，假设其长度为$m=r-k+1$，首先判断$S[k,r]$最后一次出现的位置$last$，如果$last$不存在或者$S[r-m]=S[last-m]$，那么不需要新建nq。

否则，找到最大的t满足$S[t-m+1,t]=S[k,r]$且$S[t-m]\neq S[r-m]$，然后找到最大的p满足$S[p-m+1,p]=S[k,r]$且$S[p-m]\neq S[r-m], S[p-m]\neq S[t-m]$，如下图：

如果$l\leq p-m$，我们在$r=t$时已经为它新建过nq节点了，而对于$p-m<l\leq t-m$，我们需要对它新建一个nq节点，即区间加一。

~~那么，我们就成功地把$O(nm)$优化到了$O(n^3)$！~~

考虑用后缀自动机优化这个过程。首先，对于一个串T，如果不存在两个字符$a \neq b$满足aT, bT都为S的子串，那么这个串是不用更新的，也就是说只需要考虑后缀自动机上这个节点到根节点路径上所有节点能表示的最长串。

优化

考虑用后缀自动机优化这个过程。首先，对于一个串T，如果不存在两个字符$a \neq b$满足aT, bT都为S的子串，那么这个串是不用更新的，也就是说只需要考虑后缀自动机上这个节点到根节点路径上所有节点能表示的最长串。

对于两个节点a, b，如果a的深度小于b并且a是b的祖先，还满足它们的$last$相同，那么一定有$S[last[a] - m] = S[r - m]$，所以也可以忽视掉，如图：

优化

考虑用后缀自动机优化这个过程。首先，对于一个串T，如果不存在两个字符$a \neq b$满足aT, bT都为S的子串，那么这个串是不用更新的，也就是说只需要考虑后缀自动机上这个节点到根节点路径上所有节点能表示的最长串。
对于两个节点a, b，如果a的深度小于b并且a是b的祖先，还满足它们的$last$相同，那么一定有$S[last[a] - m] = S[r - m]$，所以也可以忽视掉，如图：

还有一个问题是找到p，我们可以对后缀自动机上每个节点记录一个$diff[i]$表示上次更新的位置，那么这一次就是就是对$[diff[i] + 1, last[i] - m]$进行区间加法。

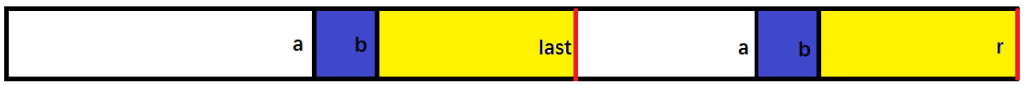
优化

考虑用后缀自动机优化这个过程。首先，对于一个串T，如果不存在两个字符$a \neq b$满足aT, bT都为S的子串，那么这个串是不用更新的，也就是说只需要考虑后缀自动机上这个节点到根节点路径上所有节点能表示的最长串。
对于两个节点a, b，如果a的深度小于b并且a是b的祖先，还满足它们的$last$相同，那么一定有$S[last[a] - m] = S[r - m]$，所以也可以忽视掉，如图：

还有一个问题是找到p，我们可以对后缀自动机上每个节点记录一个$diff[i]$表示上次更新的位置，那么这一次就是就是对$[diff[i] + 1, last[i] - m]$进行区间加法。
不难发现这个过程非常类似LCT的access过程，那么用LCT和线段树维护就好了。

优化

考虑用后缀自动机优化这个过程。首先，对于一个串T，如果不存在两个字符$a \neq b$满足aT, bT都为S的子串，那么这个串是不用更新的，也就是说只需要考虑后缀自动机上这个节点到根节点路径上所有节点能表示的最长串。
对于两个节点a, b，如果a的深度小于b并且a是b的祖先，还满足它们的$last$相同，那么一定有$S[last[a] - m] = S[r - m]$，所以也可以忽视掉，如图：



还有一个问题是找到p，我们可以对后缀自动机上每个节点记录一个$diff[i]$表示上次更新的位置，那么这一次就是就是对$[diff[i] + 1, last[i] - m]$进行区间加法。
不难发现这个过程非常类似LCT的access过程，那么用LCT和线段树维护就好了。
~~这样我们就解决了本题！~~

Part 2

刚刚的做法看似很有道理，但是...

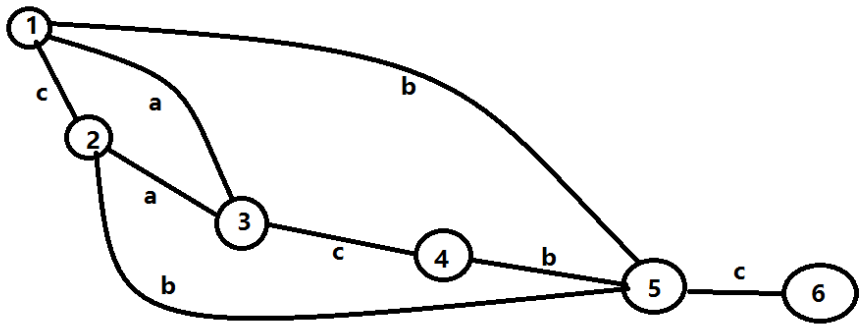
1	Wrong Answer
2	Wrong Answer
3	Wrong Answer
4	Wrong Answer
5	Wrong Answer
6	Wrong Answer
7	Wrong Answer
8	Wrong Answer
9	Wrong Answer
10	Wrong Answer
11	Wrong Answer

我们发现，上述算法计算出的结点数要比实际节点数大!

新的一个问题

如果某个子串S，存在两个不同字符a, b满足aS, bS都在原串中出现过，但是S也为原串的一个前缀，那么串S会属于一个np结点而不属于nq

比如，串$cacbc$的后缀自动机为



其中c是串的一个前缀，同时也有两个字符a,b满足ac和bc在串中出现了。那么串c就不会被新建一个结点。

对于一次区间加法，当右端点到达R时，左端点位于区间$[P,Q]$内的所有询问答案全部+1。

对于一次区间加法，当右端点到达R时，左端点位于区间$[P,Q]$内的所有询问答案全部+1。

但是如果左端点l满足$l+m-1$为当前枚举的结点k的right集合的某个元素，那么我们本不应该计算。

分析

对于一次区间加法，当右端点到达R时，左端点位于区间$[P,Q]$内的所有询问答案全部+1。

但是如果左端点l满足$l+m-1$为当前枚举的结点k的right集合的某个元素，那么我们本不应该计算。

考虑如何把这一部分的贡献减去。

分析

例如，字符串$bcacacdc$，当右端点到达8时，access到了一个结点k，其中结点k代表的串为c。那么$last[k]=6, diff[k]=2$，我们会认为，左端点l位于$[2,5]$，右端点≥ 8的所有串的SAM都要为子串c建立一个nq结点，但是，当$l=2,r=8;l=4,r=8$时c会变成一个np结点。所以，这些串就不能被统计到。
处理的方法是先把这些贡献计算进去，然后再把这个贡献减掉。

分析

例如，字符串$bcacacdc$，当右端点到达8时，access到了一个结点k，其中结点k代表的串为c。那么$last[k]=6, diff[k]=2$，我们会认为，左端点l位于$[2,5]$，右端点≥ 8的所有串的SAM都要为子串c建立一个nq结点，但是，当$l=2,r=8;l=4,r=8$时c会变成一个np结点。所以，这些串就不能被统计到。
处理的方法是先把这些贡献计算进去，然后再把这个贡献减掉。

对于一次加法，假设右端点$\geq R$，左端点位于$[P,Q]$内的所有询问答案全部+1，而如果左端点l满足$l+m-1$为当前枚举的结点k的right集合的某个元素，那么最终还需要-1。

这样的四元组(P,Q,k,R)共有$O(nlogn)$个。
(每一次区间加法会相应产生一个四元组，刚刚用LCT证明了加法次数是$nlogn$的)

统计

接下来要做的就是对于每一个询问的子串$S[l,r]$，查询这个串在多少个四元组中。
首先k所代表的最长串会以位置l开头。那么我们不妨先预处理出所有以l开头的串在后缀自动机中对应的结点，这些节点在后缀自动机的trans图中是一条从根出发的链。

统计

接下来要做的就是对于每一个询问的子串$S[l,r]$，查询这个串在多少个四元组中。
首先k所代表的最长串会以位置l开头。那么我们不妨先预处理出所有以l开头的串在后缀自动机中对应的结点，这些节点在后缀自动机的trans图中是一条从根出发的链。

同时，只有l开头的，且前驱唯一的结点才会被计算。

接下来要做的就是对于每一个询问的子串$S[l,r]$，查询这个串在多少个四元组中。
首先k所代表的最长串会以位置l开头。那么我们不妨先预处理出所有以l开头的串在后缀自动机中对应的结点，这些节点在后缀自动机的trans图中是一条从根出发的链。

同时，只有l开头的，且前驱唯一的结点才会被计算。

而后缀自动机有一个很好的性质：如果一个串T的前驱唯一并且T不是一个前缀，那么它会与它的孩子合并，那么，代表这个串的结点的mx一定要大于$|T|$。

我们按照从小到大的顺序枚举所有以l开头的串，那么随着串长度的增加，串的出现次数一定是单调不增的，这也就说明它的前驱种类是单调不增的。

我们按照从小到大的顺序枚举所有以l开头的串，那么随着串长度的增加，串的出现次数一定是单调不增的，这也就说明它的前驱种类是单调不增的。

所以，满足前驱不唯一条件的一定是一段前缀，也就是后缀自动机的trans图上从root出发的一条路径。

考虑使用扫描线，对于四元组(P,Q,k,R)在位置P加入，在位置$Q+1$删除。
每次我们需要给一个结点k添加一个数R，然后对于每一个询问，查询后缀自动机的trans图上一条从root出发的路径上$\leq r$的数的个数。

考虑使用扫描线，对于四元组(P,Q,k,R)在位置P加入，在位置$Q+1$删除。
每次我们需要给一个结点k添加一个数R，然后对于每一个询问，查询后缀自动机的trans图上一条从root出发的路径上$\leq r$的数的个数。
但是后缀自动机的trans图是一张图，对于图信息的维护非常不方便，我们考虑把它转成树的模型。

构树

在证明后缀自动机线性状态数的时候，我们引入了后缀自动机的trans图的生成树。

构树

在证明后缀自动机线性状态数的时候，我们引入了后缀自动机的trans图的生成树。

同样我们可以建立SAM的一棵生成树。每一个后缀可以唯一对应一条非树边。

构树

在证明后缀自动机线性状态数的时候，我们引入了后缀自动机的trans图的生成树。

同样我们可以建立SAM的一棵生成树。每一个后缀可以唯一对应一条非树边。

后缀自动机还有一个小性质：trans图上，有边到达点k的所有点的$[mi_k, mx_k]$互不相交，且它们的并恰好组成一段连续的区间，其中mx最大的元素恰好为$mx_k - 1$。

求解

求解

同时我们注意到，只有mx值为$mx_k - 1$的点连过来的边才是有效的。所以，构造完生成树后，一个后缀要查询的最大长度就是这个后缀在trans图上的路径在第一次遇到非树边之前的路径长度。

求解

同时我们注意到，只有mx值为mx_k-1的点连过来的边才是有效的。所以，构造完生成树后，一个后缀要查询的最大长度就是这个后缀在trans图上的路径在第一次遇到非树边之前的路径长度。

这个是满足单调性的，我们可以用哈希把生成树中root到每个结点的路径的哈希值记录下来，每次二分查找最大查询长度。

这样我们就把操作转化为了：往树中某个结点插入一个数、询问一个点到根的路径中小于等于给定值的数的个数。

求解

同时我们注意到，只有mx值为mx_k-1的点连过来的边才是有效的。所以，构造完生成树后，一个后缀要查询的最大长度就是这个后缀在trans图上的路径在第一次遇到非树边之前的路径长度。
这个是满足单调性的，我们可以用哈希把生成树中root到每个结点的路径的哈希值记录下来，每次二分查找最大查询长度。
这样我们就把操作转化为了：往树中某个结点插入一个数、询问一个点到根的路径中小于等于给定值的数的个数。

这是一个二维的问题，可以用树套树维护。

这样我们就解决了np和nq重复的问题。

这样我们就解决了np和nq重复的问题。

然后就可以AC本题啦！

0	Compilation Success	null ms	null KB	0
1	Accepted	4 ms	75120 KB	10
2	Accepted	12 ms	75100 KB	10
3	Accepted	32 ms	77956 KB	10
4	Accepted	28 ms	77948 KB	10
5	Accepted	664 ms	117632 KB	10
6	Accepted	588 ms	106876 KB	10
7	Accepted	724 ms	109268 KB	10
8	Accepted	720 ms	109248 KB	10
9	Accepted	836 ms	126988 KB	10
10	Accepted	844 ms	125532 KB	10
11	Accepted	1020 ms	138304 KB	0

之前的做法的时间复杂度高达 $O(n\log^3 n + m\log^2 n)$，并且在问题解决时用到了 SAM 生成树的相关知识，~~太难了根本无法理解~~非常复杂。

优化

之前的做法的时间复杂度高达 $O(n\log^3 n + m\log^2 n)$，并且在问题解决时用到了 SAM 生成树的相关知识，~~太难了根本无法理解~~非常复杂。

注意到时间复杂度瓶颈在于第二部分对 np 节点被误算为 nq 的处理。考虑回归到一开始的问题：统计所有询问串的前缀 T 满足存在两个不同的字符 a, b 使得在询问串中同时出现过 aT 和 bT。

之前的做法的时间复杂度高达 $O(n\log^3 n + m\log^2 n)$，并且在问题解决时用到了 SAM 生成树的相关知识，~~太难了根本无法理解~~非常复杂。

注意到时间复杂度瓶颈在于第二部分对 np 节点被误算为 nq 的处理。考虑回归到一开始的问题：统计所有询问串的前缀 T 满足存在两个不同的字符 a, b 使得在询问串中同时出现过 aT 和 bT。

设询问串为 $S[L, R]$，**那么我们可以注意到若前缀** $T = S[L, p](L < p)$ **满足上面的条件，那么** $T = S[L, p-1]$ **也满足条件。**

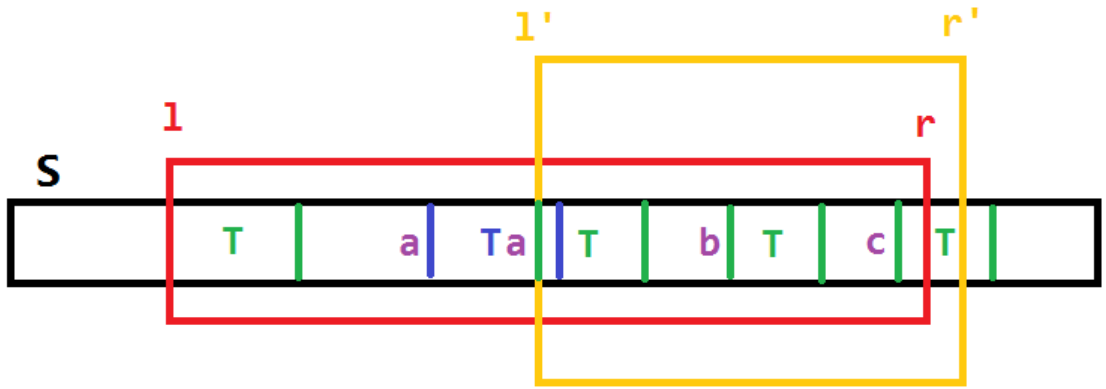
之前的做法的时间复杂度高达 $O(n\log^3 n + m\log^2 n)$，并且在问题解决时用到了 SAM 生成树的相关知识，~~太难了根本无法理解~~非常复杂。

注意到时间复杂度瓶颈在于第二部分对 np 节点被误算为 nq 的处理。考虑回归到一开始的问题：统计所有询问串的前缀 T 满足存在两个不同的字符 a, b 使得在询问串中同时出现过 aT 和 bT。

设询问串为 $S[L, R]$，**那么我们可以注意到若前缀** $T = S[L, p](L < p)$ **满足上面的条件，那么** $T = S[L, p-1]$ **也满足条件。**

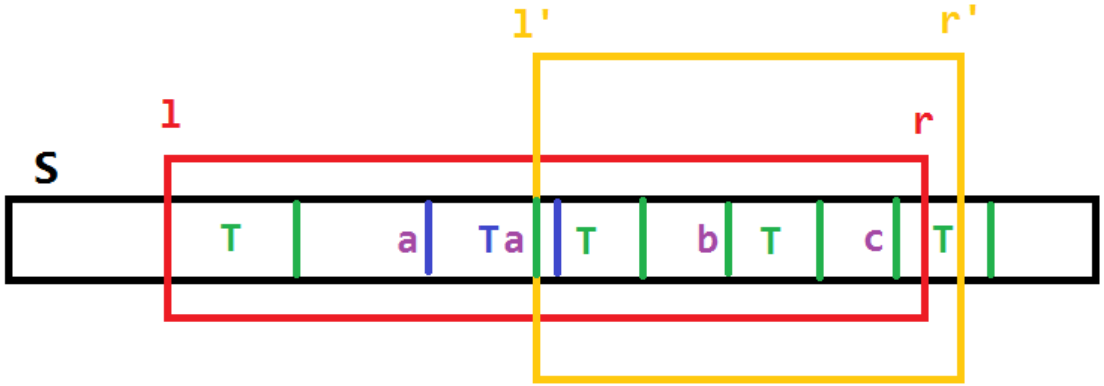
如果我们能够快速判定前缀 $T = S[L, p]$ 是否满足条件，就可以通过简单的二分来解决问题了。

SA做法



当确定了前缀 $T = S[L, p]$ 后，假设 $aT = S[l_1 - 1, r_1], bT = S[l_2 - 1, r_2]$，我们就可以确定 l_1, l_2 的范围 $[s, t]$。

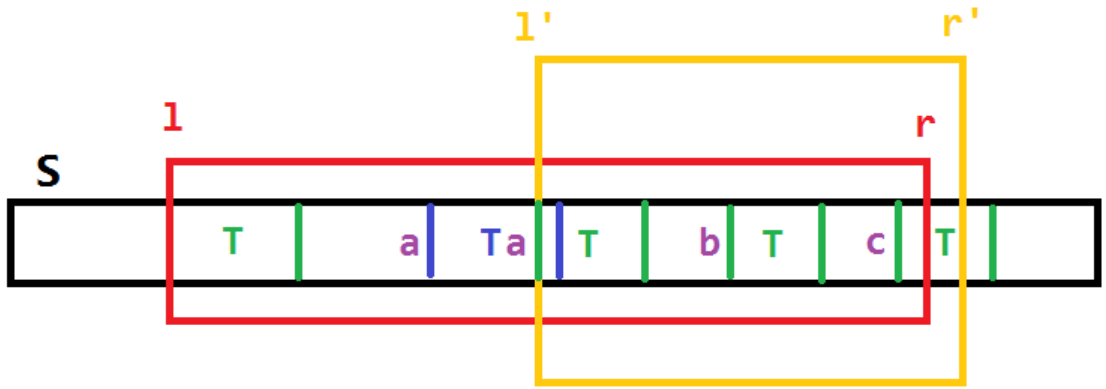
SA做法



当确定了前缀 $T = S[L, p]$ 后，假设 $aT = S[l_1 - 1, r_1], bT = S[l_2 - 1, r_2]$，我们就可以确定 l_1, l_2 的范围 $[s, t]$。

而我们判定是否存在 aS 和 bS，实际上就是判定**当左端点在 $[s, t]$ 内时，是否存在两个原串的后缀满足 $S[L, R]$ 的 LCP 不小于 $p - L + 1$，且这两个位置的前一个字符不同。**

SA做法

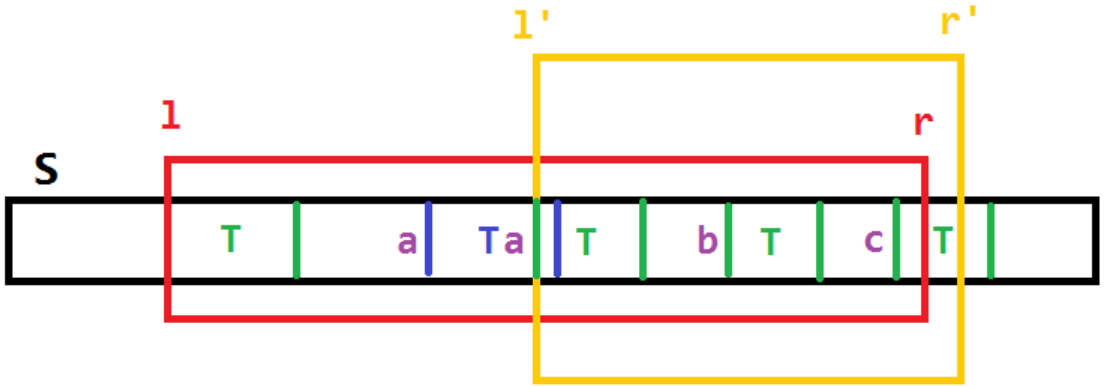


当确定了前缀 $T = S[L, p]$ 后，假设 $aT = S[l_1 - 1, r_1], bT = S[l_2 - 1, r_2]$，我们就可以确定 l_1, l_2 的范围 $[s, t]$。

而我们判定是否存在 aS 和 bS，实际上就是判定**当左端点在 $[s, t]$ 内时，是否存在两个原串的后缀满足 $S[L, R]$ 的 LCP 不小于 $p - L + 1$，且这两个位置的前一个字符不同。**

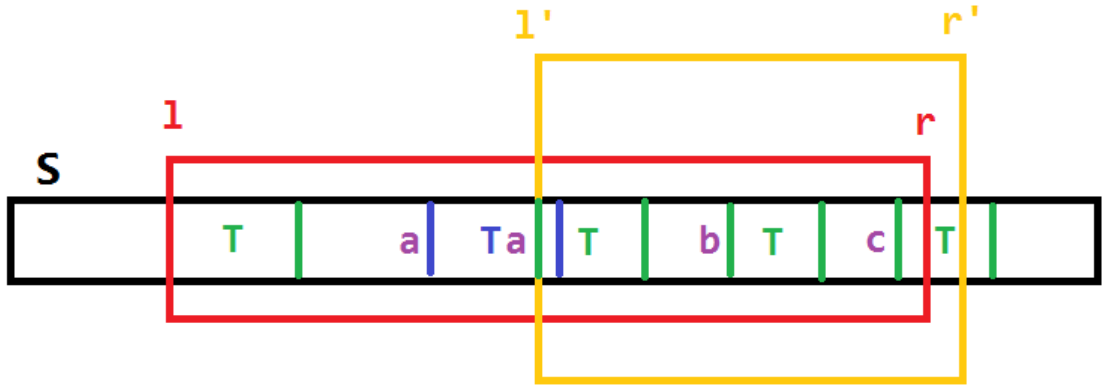
如图所示，当询问串为 $S[l, r]$ 时，可以找到 aT 和 bT，因为 LCP 的长度满足条件，同时右端点又不超过 r。但询问串为 $S[l', r']$ 时，就不能找到 bT 和 cT，因为 cT 的右端点越过了 r'。

SA做法



我们可以预处理出 S 的后缀数组，并按照 $height$ 从大到小进行合并，并部分可持久化这个过程。每次询问的就是某个时刻位置 L 的联通块中位置在 $[s,t]$ 区间内的所有数的值是否都相同。

SA做法



我们可以预处理出 S 的后缀数组，并按照 $height$ 从大到小进行合并，并部分可持久化这个过程。每次询问的就是某个时刻位置 L 的联通块中位置在 $[s,t]$ 区间内的所有数的值是否都相同。使用可持久化的线段树启发式合并即可，这部分的预处理的时间复杂度为 $O(n\log n)$，于是第二部分的时间复杂度就被优化到了 $O(m\log^2 n)$。

SA做法

已经写了后缀自动机，可不可以不写后缀数组啊？

SA做法

已经写了后缀自动机，可不可以不写后缀数组啊？

SA做法

已经写了后缀自动机，可不可以不写后缀数组啊？

~~当然是选择用SAM求SA啦~~。

后缀树做法

当然可以啦！

后缀树做法

当然可以啦！
我们知道正串的后缀树就是反串的后缀自动机的 $parent$ 树，于是我们建立 S 的反串，也就是输入的原串的后缀自动机。我们在后缀树上进行线段树合并，就能得到和 SA 一样的效果。每次只要二分到后缀树上最浅的深度 $\geq p-L+1$ 的祖先，并在对应的线段树中查询即可。

后缀树做法

当然可以啦！
我们知道正串的后缀树就是反串的后缀自动机的 $parent$ 树，于是我们建立 S 的反串，也就是输入的原串的后缀自动机。我们在后缀树上进行线段树合并，就能得到和 SA 一样的效果。每次只要二分到后缀树上最浅的深度 $\geq p-L+1$ 的祖先，并在对应的线段树中查询即可。
实际上和 SA 做法并没有什么本质不同，但是你不用再写一个 SA 了！

后缀树做法

已经按输入的反串顺序建好了 SAM，可不可以不重新加一遍啊？

后缀树做法

已经按输入的反串顺序建好了 SAM，可不可以不重新加一遍啊？

后缀树做法

已经按输入的反串顺序建好了 SAM，可不可以不重新加一遍啊？

SAM做法

当然可以啦！

当然可以啦！
考虑我们二分的过程实际上就是判定在一个串在某个区间 $[l,r]$ 内是否存在不同的前驱。于是我们只要在 $parent$ 树上找到这个串对应点，就只要判断在 $[l,r]$ 中是否有元素来自不同子树。于是只要同样对 $right$ 集合做启发式合并，再根据一个满足条件的 aT 找到一个满足条件的子树，判断是否所有的 $[l,r]$ 之间的位置是否都出现在这棵子树中即可。
时间复杂度同样是 $O(m\log^2 n)$。

然后就可以更快 AC 本题啦！
似乎由于 $m \geq n$ 并没有快很多的样子。

然后就可以更快 AC 本题啦！

似乎由于 $m \geq n$ 并没有快很多的样子。

#	状态	时间	空间	得分	评测信息
0	Compilation Success	null ms	null KB	0	
1	Accepted	25 ms	56568 KB	10	
2	Accepted	29 ms	56440 KB	10	
3	Accepted	44 ms	59204 KB	10	
4	Accepted	35 ms	59076 KB	10	
5	Accepted	265 ms	127780 KB	10	
6	Accepted	815 ms	122188 KB	10	
7	Accepted	483 ms	133160 KB	10	
8	Accepted	428 ms	133296 KB	10	
9	Accepted	964 ms	138032 KB	10	
10	Accepted	693 ms	129944 KB	9	
11	Accepted	430 ms	98472 KB	1	

启发式合并

(码码码) 这个第一部分的 LCT 咋这么难写啊，可不可以弃疗啊?

启发式合并

(码码码) 这个第一部分的 LCT 咋这么难写啊，可不可以弃疗啊?

启发式合并

(码码码) 这个第一部分的 LCT 咋这么难写啊，可不可以弃疗啊？

~~这都写不出来听说你会 LCT~~。

启发式合并

我们换一种思路。考虑对每个节点的 $right$ 集合排序，那么考虑之前用 LCT 维护 $diff$ 的过程。若相邻的位置来自不同的子树，那么就会将 $diff$ 后移一次。

启发式合并

我们换一种思路。考虑对每个节点的 $right$ 集合排序，那么考虑之前用 LCT 维护 $diff$ 的过程。若相邻的位置来自不同的子树，那么就会将 $diff$ 后移一次。
于是，对于排好序的 $right$ 集合，当相邻的两个不同时，就会在某个时刻 T 对区间 $[P,Q]$ 进行区间加一。

启发式合并

我们换一种思路。考虑对每个节点的 $right$ 集合排序，那么考虑之前用 LCT 维护 $diff$ 的过程。若相邻的位置来自不同的子树，那么就会将 $diff$ 后移一次。

于是，对于排好序的 $right$ 集合，当相邻的两个不同时，就会在某个时刻 T 对区间 $[P,Q]$ 进行区间加一。

而我们如何维护这样的过程呢？考虑从 $right$ 集合最大的子树继承 $right$ 集合，这时没有任何贡献。那么之后每个节点暴力插入最多产生新的两段，因此加的总次数不会超过 $2n\log n$。于是暴力插入就可以了。由于时间复杂度瓶颈并不在这里，因此只要用 set 或线段树进行启发式合并即可。

这一部分的时间复杂度仍是 $O(n\log^2 n)$，但是省去了 LCT 部分的操作。

启发式合并

我们换一种思路。考虑对每个节点的 $right$ 集合排序，那么考虑之前用 LCT 维护 $diff$ 的过程。若相邻的位置来自不同的子树，那么就会将 $diff$ 后移一次。

于是，对于排好序的 $right$ 集合，当相邻的两个不同时，就会在某个时刻 T 对区间 $[P,Q]$ 进行区间加一。

而我们如何维护这样的过程呢？考虑从 $right$ 集合最大的子树继承 $right$ 集合，这时没有任何贡献。那么之后每个节点暴力插入最多产生新的两段，因此加的总次数不会超过 $2n\log n$。于是暴力插入就可以了。由于时间复杂度瓶颈并不在这里，因此只要用 set 或线段树进行启发式合并即可。

这一部分的时间复杂度仍是 $O(n\log^2 n)$，但是省去了 LCT 部分的操作。

于是我们震惊地发现，只用一个 SAM 和线段树，就可以解决这个看起来非常困难的问题了！

前置技能

SAM，LCT，线段树
SAM生成树，启发式合并，二分+哈希，倍增
扫描线，树套树/CDQ分治+线段树

前置技能

SAM，~~LCT~~，线段树

~~SAM生成树~~，启发式合并，~~二分+哈希~~，~~倍增~~

扫描线，~~树套树/CDQ分治~~，~~线段树~~

Thanks!

~~我们可以欣喜地发现，这道题是一道简单的省选级别的题目！~~

总结

总结

- 这是一道考察后缀自动机的题目。

总结

- 这是一道考察后缀自动机的题目。
- 初步思路源于去年集训队论文中《基因组重构》的方法。使用LCT进行维护。

总结

- 这是一道考察后缀自动机的题目。
- 初步思路源于去年集训队论文中《基因组重构》的方法。使用LCT进行维护。

- 在研究的过程中发现启发式合并可以替代LCT。

总结

- 这是一道考察后缀自动机的题目。
- 初步思路源于去年集训队论文中《基因组重构》的方法。使用LCT进行维护。

- 在研究的过程中发现启发式合并可以替代LCT。

- 利用后缀自动机的一些性质解决了np和nq重复的问题。

总结

- 这是一道考察后缀自动机的题目。
- 初步思路源于去年集训队论文中《基因组重构》的方法。使用LCT进行维护。

- 在研究的过程中发现启发式合并可以替代LCT。

- 利用后缀自动机的一些性质解决了np和nq重复的问题。
- 朱老大利用前驱的单调性，简化了做法。

其它

其它

本题的字符串，字符集大小是线性的。那么后缀自动机的边可以用map存。
需要注意的是，用map实现的SAM的复杂度是和字符集大小无关的$O(nlogn)$。
命题来源：在探究后缀自动机复杂度的过程中，突然想出的一道题。发现可以用2017年论文中的方法进行优化。
在探究过程中发现，论文中的LCT可以用启发式合并代替，并有着相同的复杂度。
为了让本次营员交流更加成功，特邀请了加强和实现了本题的两个同学~~(du liu)~~。

感谢

感谢$w \times h$的验题，朱老大对标准算法的改进，fsf指出本题题面的错误。

感谢CCF提供这次营员交流的机会。

感谢TUOJ提供了放题的平台。

感谢大家的聆听。