简单字符串

洛水·锦依卫

August 18, 2019

KMP

众所周知，KMP 是最普通的单模式串匹配算法，其核心在于已经匹配过的部分无需继续匹配。

KMP

众所周知，KMP 是最普通的单模式串匹配算法，其核心在于已经匹配过的部分无需继续匹配。

大致可以如此理解：对于字符串 s，若当前已经匹配到了前缀位置 i，那么对于这个前缀的每个后缀自然也被匹配到了。此时若 $i+1$ 失配，若有某个已匹配后缀等同于某个前缀，则此前缀也已经被匹配到，那就无需跳到文本串与模式串开头重新匹配，只需要好好利用已匹配部分即可。

KMP

换句话说，根据这个思想，我们每次只需要跳到最长的，与后缀集合有交的前缀即可。而 nx 数组就记录了这个最长前缀的长度。

KMP

换句话说，根据这个思想，我们每次只需要跳到最长的，与后缀集合有交的前缀即可。而 nx 数组就记录了这个最长前缀的长度。

所以说 KMP 算法最重要的部分其实也就在于求 nx 了，之后的大家都会。

KMP

换句话说，根据这个思想，我们每次只需要跳到最长的，与后缀集合有交的前缀即可。而 nx 数组就记录了这个最长前缀的长度。

所以说 KMP 算法最重要的部分其实也就在于求 nx 了，之后的大家都会。

nx 的求法有两种差不多的理解方式，一种是普通 KMP 的实现方法，另一种则可以通过 AC 自动机的构造方法来理解。

KMP

KMP 实现 $next$ 数组的方式很简单，若要找到新添的字符对应的最长前后缀交，则在前后缀去掉这个字符时依然相等。那么我们找到之前的最后一个字符，不断寻找与它的后缀相等的前缀，并尝试扩展即可。这样的话均摊复杂度 $O(n)$ 。

HNOI 2008 *GT* 考试

现有一串长度为 m 的数字串，请问有多少种长度为 n 的数字串满足 m 不为其子串。答案对 k 取余。

$n \leq 10^9, m \leq 20, k \leq 1000$

设 $f_{i,j}$ 为当前已经到位置 i，已匹配前缀长度为 j，枚举下一位是 $0-9$ 做转移至 f_{i+1} 可得到暴力分。

设 $f_{i,j}$ 为当前已经到位置 i ，已匹配前缀长度为 j，枚举下一位是 $0-9$ 做转移至 f_{i+1} 可得到暴力分。

由于第一维可滚，设 $g_{i,j}$ 为当前已匹配到长度 i 的前缀，有多少个数字能使其长度变成 j，求此数组可通过 KMP 实现，我们发现可以将其写成矩乘形式，即：$f_i = \sum f_j * g_{j,i}$ 。直接快速幂优化即可。

NOI 2014 动物园

设 num_i 为前缀 i 所拥有的，与其相同长度后缀相等且不相交的的前缀个数。
T 组询问，每组询问给出一字符串，求 $\prod num_i + 1$ 。

设字符串长度为 L 。
$T \leq 5, L \leq 10^6$

第一眼应该都觉得是倍增吧？处理出在 nx 树上的幂次祖先，log 跳到小于等于 $\frac{i}{2}$ 的位置，加上深度的贡献。复杂度为 $O(Tnlog)$ 。

第一眼应该都觉得是倍增吧？处理出在 nx 树上的幂次祖先，log 跳到小于等于 $\frac{i}{2}$ 的位置，加上深度的贡献。复杂度为 $O(Tnlog)$ 。

这个写法理论上复杂度问题不大，刚好卡过，但是基于常数问题有小部分人可能会被卡掉。所以我们有两种使自己更稳的方法，一是卡常，二是优化复杂度。

先谈谈卡常，正常的倍增写法是 $f_{i,j}$ 表示 i 的 2^j 级祖先。如果我们改为 $f_{j,i}$ 来表示就可以做到快上三倍。

先谈谈卡常，正常的倍增写法是 $f_{i,j}$ 表示 i 的 2^j 级祖先。如果我们改为 $f_{j,i}$ 来表示就可以做到快上三倍。

我也不知道为什么呀……这个很玄学

至于优化复杂度其实很简单，由于这道题的本质是在 nx 树上找到对应节点计算贡献，那么我们其实可以考虑重复一遍建 nx 的过程来找节点，只需要注意跳 nx 的时候附带上必须 $\leq \frac{i}{2}$ 的条件即可。

HNOI 2019 *JOJO*

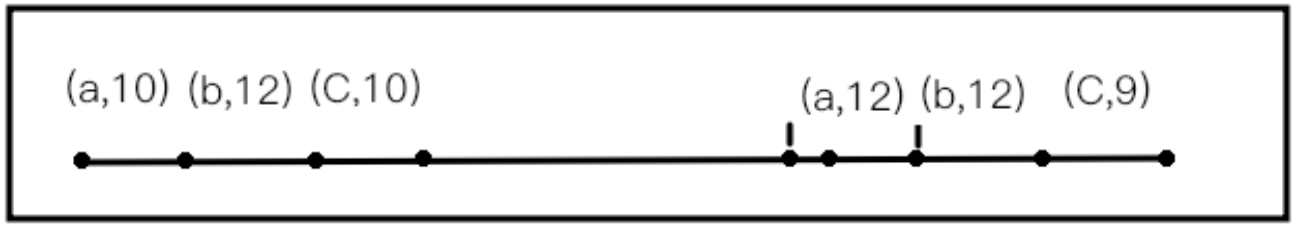
你有一个字符串，开始时字符串为空。接下来有 n 个操作，共两种，第一种是在当前串结尾加入 x 个字符 c，且保证 c 不与当前串结尾字符相同。第二种是将字符串还原回第 x 个操作之后的状态。每次操作后询问每个位置的 nx 数组的值之和。

$n \leq 10^5, x \leq 10^4$

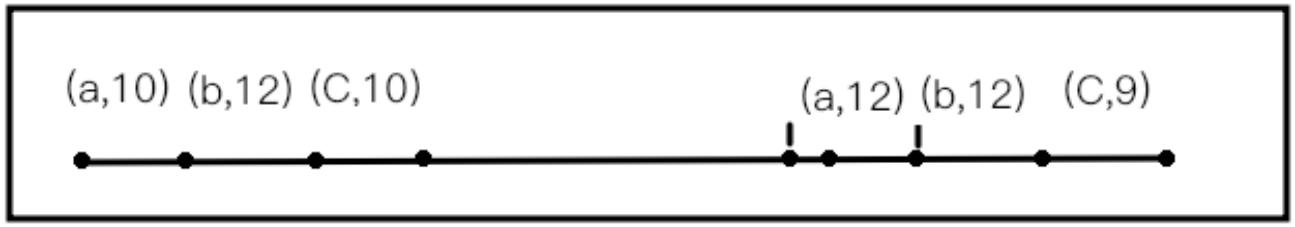
对于每次第一种操作加入的字符，我们将其看作一个整体，可以称其为字段，一个字段拥有字符与长度两种属性。

对于每次第一种操作加入的字符，我们将其看作一个整体，可以称其为字段，一个字段拥有字符与长度两种属性。

先考虑一个 50 分做法（虽然说是 50 分，但本题数据水，实际上可以 A 掉）：当我们在结尾加入一个字符时，回想一下跳 KMP 的过程：不断跳前一位的 nx ，直到当前位置的后一个字符与加入字符相同。

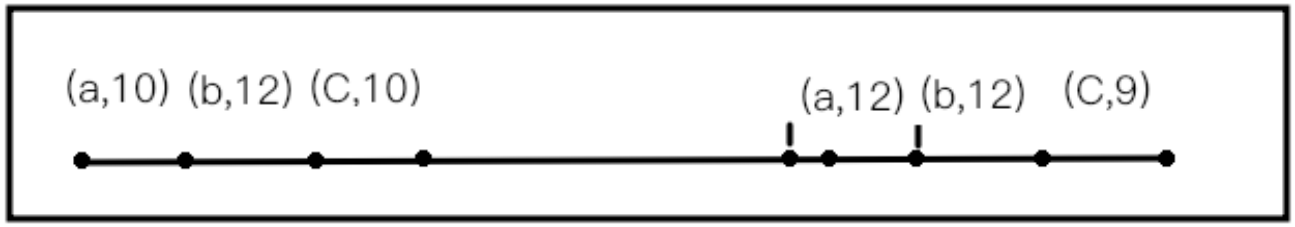


那么由于每次加入的字段都与前面的字符不同，则我们发现，对于一对相同的前后缀，删掉开头结尾的第一个字段，中间的都是完整的字段。那么我们可以将一个字段视作一个新的字符进行 KMP ，同时特别的，对于第一个字段，我们将所有与它字符相同且长度大于它的字段视作相同字段。

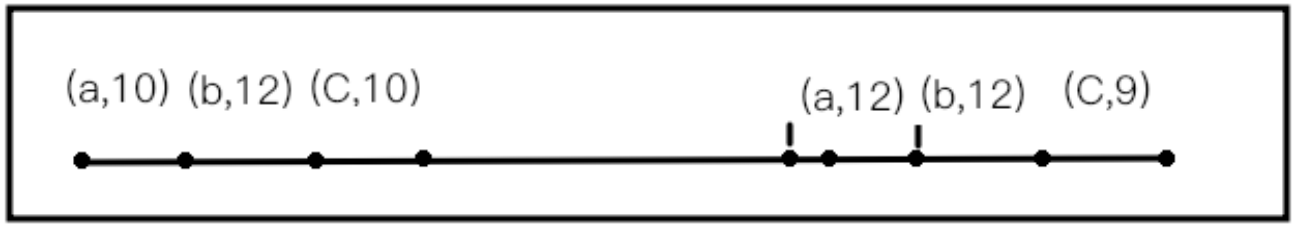


那么由于每次加入的字段都与前面的字符不同，则我们发现，对于一对相同的前后缀，删掉开头结尾的第一个字段，中间的都是完整的字段。那么我们可以将一个字段视作一个新的字符进行 KMP ，同时特别的，对于第一个字段，我们将所有与它字符相同且长度大于它的字段视作相同字段。

那么每次新加入一个字段，我们只需要不断跳 nx 并计算答案。



虽然此算法能通过此题，但毕竟复杂度不正确，因为 KMP 跳数组的 $O(n)$ 是均摊意义下的，若有回溯操作并刻意构造就能够完美卡掉它。那么考虑令跳 KMP 的过程复杂度正确。



虽然此算法能通过此题，但毕竟复杂度不正确，因为 KMP 跳数组的 $O(n)$ 是均摊意义下的，若有回溯操作并刻意构造就能够完美卡掉它。那么考虑令跳 KMP 的过程复杂度正确。

可以考虑一个平时由于复杂度均摊而完全不会考虑的优化：循环节。对于跳 nx，假设当前在位置 i，若 $nx_i < \frac{i}{2}$，则跳 nx 会使长度减少到一半以下。但如果 $nx_i > i/2$，则可能导致长度只会减少一点点，从而复杂度错误。

但是，如果 $nx_i > i/2$ ，它就会产生至少两个循环节！那么我们只需要加上一个判断：若当前前缀 i 存在循环节，先判断末尾循环节是否满足要求，然后调试第一个循环节即可。

但是，如果 $nx_i > i/2$ ，它就会产生至少两个循环节！那么我们只需要加上一个判断：若当前前缀 i 存在循环节，先判断末尾循环节是否满足要求，然后调试第一个循环节即可。

这样的话每次长度必定缩短一半以上，则跳 KMP 的复杂度上限优化为每次 $O(log(n))$ ，总复杂度 $O(nlogn)$。

AC 自动机

第二种简单算法自然就是多模式匹配的 AC 自动机了。（我不认为最开始听到这个名字的时候只有我想象成自动 AC 机）

AC 自动机

第二种简单算法自然就是多模式匹配的 AC 自动机了。（我不认为最开始听到这个名字的时候只有我想象成自动 AC 机）

AC 自动机的思想十分简单：想一下我们平时暴力匹配，是枚举每个子串。第一想法是枚举字串长度或左端点再暴力枚举子串。但是我们还有另一种暴力枚举字串的方法，那就是枚举前缀，再枚举前缀的后缀。AC 自动机就是基于这个思想。所以 AC 自动机上的失配边有两种讲法，一是 $fail$ 边，二是后缀节点。

AC 自动机

则我们构建 AC 自动机的时候，可以考虑对 $Trie$ 树上每个节点处理出其后缀节点。后缀节点的定义为：当前路径构成的字符串的非本身最长后缀在 $Trie$ 上走到的位置。那么我们在匹配文本串的时候，只需要在 $Trie$ 上走，然后不断跳后缀节点检查当前前缀的每一个后缀即可。

NOI 2011 阿狸的打字机

现有一空串，接下来有 n 个操作，一是在串尾加入小写字母，二是将当前串记录，三是删除当前串末尾字母。接下来 m 个询问，每次询问第 x 个记录的串在第 y 个记录的串中出现了多少次。

$n, m \leq 10^5$

经典题目了，不难想到先建 AC 自动机。具体构建方法是记录当前串与上一次记录的串的重复长度以优化复杂度。

经典题目了，不难想到先建 AC 自动机。具体构建方法是记录当前串与上一次记录的串的重复长度以优化复杂度。

然后我们发现对于询问的文本串其实已经在 $Trie$ 里了，所以对于暴力来说，实际上只需要将文本串路径上的每个节点都跳一下失配边并记录答案即可。

由于失配边必定会构成一颗树，所以我们可以反过来看：对于模式串，查询失配树的子树内有多少个节点是文本串可以跳到的。那么我们只需要用树状数组结合 dfs 序动态查询子树和，并将询问挂在文本串末尾处，然后 dfs 遍历 $Trie$ 树以依次加入每个文本串，并在每个文本串末尾查询对应询问的模式串的子树和即可。

POI 2000 病毒

有 n 个 01 串，问是否能构造出无限长的文本串使得任意一个 01 串都不为此串的子串。设 len 为 n 个 01 串的长度之和。

$n \leq 2000, len \leq 30000$

经典题目。讲白了就是问你 AC 自动机上是否存在一个节点均无法匹配任意一个模式串的环。建完 AC 自动机直接 dfs 即可。

JSOI 2007 文本生成器

给出 n 个模式串，皆有由大写字母组成。问有多少种由大写字母组成的，长度为 m 的文本串满足其中至少有一个模式串。模式串不会超过 m 。

$n \leq 60, m \leq 100$

经典题目。不难想到将条件变成“所有可能的情况减去不含模式串的情况”。那么直接设 $f_{i,j}$ 表示文本串在 AC 自动机上走到 i 节点，此时长度为 j 且不含模式串的方案数，然后 DP 即可。

CF86C Geneticengineering

给出 m 个模式串 s，问有多少种长度为 n 的字符串，使得对于任意一个位置 i，都满足存在一个区间 $l \leq i \leq r$，使得这段区间代表的子串与某一个模式串相等。

$n \leq 1000, m \leq 10, |s| \leq 10$

先建立 AC 自动机，然后考虑在 AC 自动机上 dp。

先建立 AC 自动机，然后考虑在 AC 自动机上 dp。

设 $f_{len,x,left}$ 表示还剩下长度 i 未构造，当前在节点 x ，且由包括 i 位置往前数 $left$ 位都没有匹配任意一个模式串。

先建立 AC 自动机，然后考虑在 AC 自动机上 dp。

设 $f_{len,x,left}$ 表示还剩下长度 i 未构造，当前在节点 x ，且由包括 i 位置往前数 $left$ 位都没有匹配任意一个模式串。

对于每个节点，处理出所有后缀节点最长的那个可匹配串的长度，设为 $maxlen$。设 v 为自动机上下一节点，则如果 $maxlen_v \geq left+1$ ，前面的 $left$ 位就都能匹配到了，那么转移到 $f_{len-1,v,0}$。反之，则转移到 $f_{len+1,v,left+1}$。过程可以用记忆化搜索实现。

如果当前状态的 $left \geq Max_{|s|}$ ，那么可以直接返回 0。当 $len = 0$ 时，如果 $left = 0$ 则返回 1，反之返回 0。

CF696D Legen...

有 n 个模式串，每个模式串都有一个价值 v_i ，即第 i 个模式串的贡献为在文本串中的出现次数乘 v_i 。要求构造长度为 L 的文本串，使得其总贡献最大。

$n \leq 200, v_i \leq 100, L \leq 10^{14}$，模式串长度之和 ≤ 200

时限 $6s$ 。

暴力做法是设 $f_{i,j}$ 为文本串构造到第 i 位，当前在 AC 自动机的节点 j 的最大贡献。
但是显然，L 太大了。

暴力做法是设 $f_{i,j}$ 为文本串构造到第 i 位，当前在 AC 自动机的节点 j 的最大贡献。
但是显然，L 太大了。

不过我们发现 n 和模式串总长度特别小，那就有突破口了。

首先，第一维显然可滚。那么不难发现，设 $g_{i,j,k}$ 为在自动机上的 i 节点走 j 布到达 k 节点的最大贡献，则 g 数组一定是固定的。

首先，第一维显然可滚。那么不难发现，设 $g_{i,j,k}$ 为在自动机上的 i 节点走 j 布到达 k 节点的最大贡献，则 g 数组一定是固定的。

那么我们考虑将 L 二进制拆分，对于接下来的 2^i 步，我们可以利用倍增来求出 g 数组。

首先，第一维显然可滚。那么不难发现，设 $g_{i,j,k}$ 为在自动机上的 i 节点走 j 布到达 k 节点的最大贡献，则 g 数组一定是固定的。

那么我们考虑将 L 二进制拆分，对于接下来的 2^i 步，我们可以利用倍增来求出 g 数组。

具体怎么求呢？只需要使用 $floyed$ 算法，枚举松弛点即可从 2^{i-1} 次方的答案转移至 2^i 次方。复杂度 $O(n^3log)$

后缀数组

后缀数组其实就是将每个后缀形式化处理的一个东西。由于字符串的每个子串都可以表示为某个后缀的前缀，所以后缀数组在处理各种子串上都很有一手。

后缀数组

后缀数组其实就是将每个后缀形式化处理的一个东西。由于字符串的每个子串都可以表示为某个后缀的前缀，所以后缀数组在处理各种子串上都很有一手。

后缀数组依赖于后缀排序，一般使用倍增加上基数排序实现。在处理之后，我们会得到三个数组：$rk, sa, height$，rk_i 代表从 i 位置开始的后缀的排名，sa_i 代表排名第 i 的后缀所在的位置，$height_i$ 代表排序后第 i 个后缀与第 $i-1$ 个后缀的重合前缀长度。由于已经排过了序，所以 $height$ 数组用处很多。

CF271D GoodSubstrings

给定小写字符串 s，定义若干小写字母为特殊字母，请求出本质不同且包含特殊字母小于 k 个的子串个数。

$0 \leq k \leq |s| \leq 1500$

最粗浅的题目，放出来给忘却或者没学后缀数组的人一些回忆。

最粗浅的题目，放出来给忘却或者没学后缀数组的人一些回忆。

首先发现可以 $|s|^2$ 判断每个区间，那事情就很好办了。先记录 sum_i 表示前缀 i 中特殊字母个数，然后后缀排序，求出 $height$ 数组，按照后缀顺序固定左端点 L 为后缀开头，右端点则从 $L+height_i$ 开始枚举，因为后缀 sa_i 和 sa_{i-1} 之间相同的部分已经算过了。

CF427D Match&Catch

给出串 $s1, s2$ ， 要求找到最短的子串 t 使得 t 在 $s1, s2$ 中均只出现过一次。

$|s1|, |s2| \leq 5000$

后缀数组常用手段：将所有串拼起来，并在不同的串之间插入特殊符号。这样就能做到排序后一般不会有跨越不同字符串的后缀出现。

后缀数组常用手段：将所有串拼起来，并在不同的串之间插入特殊符号。这样就能做到排序后一般不会有跨越不同字符串的后缀出现。

那么将 $s1, s2$ 拼接在一起并进行后缀排序，枚举子串 t 的长度 len，长度为 len 的子串满足题目要求当且仅当存在 sa_i 与 sa_{i-1} 分别来自两个字符串，且 $height_i \geq len$ ，且 $height_{i-1}, heighti + 1 \leq len$。

CF123D String

给定字符串 s，对于每一个本质不同的子串 t，设 cnt_t 为它在 s 里的出现次数，求：

$$\sum \frac{(cnt_t + 1) * t}{2}$$

$|s| \leq 10^5$

首先转换思路：怎么求一个子串在原串中出现了多少次？我们考虑换个更明显的模型来思考：若将所有后缀插进一颗 $Trie$ 里得到后缀树会怎么样？那么对于从根节点出发，到每一个节点的路径，形成的都是原串的不重复子串。至于一个子串出现的次数，不就是后缀树上对应节点子树内的叶子个数吗？后缀数组求出了相邻两个后缀的 $height$ ，其实也就是求出了它们在后缀树上的 LCA 深度。

考虑后缀排序得到的数组的性质：对于一段排序后的连续后缀来说，它们共同的 LCA 为这段区间内 $height$ 的最小值。鉴于这些七七八八的性质在后缀树上的体现，我们可以改变所求的方向，也就是后缀树上每个节点的贡献。不难发现，对于后缀树上的一段链，若链中间（不包括端点）每个节点的度数都 ≤ 2 ，那么这条链的贡献相等。

考虑用单调栈来维护这些东西。我们维护一个 $height$ 单调递增的栈，这样就可以保证若 $height$ 变小，那么之前 $\geq height_i$ 的 $height$ 都会被计算贡献。则我们只需要进行如下步骤。

若当前点 $\geq$ 单调栈内最大值，则压入栈顶。反之进行弹栈，弹栈操作如下：

- 找到栈顶元素的 $height$
- 计算从 $height$ 往上有多少节点贡献相同，将其乘以叶子节点数的相应贡献
- 将栈顶元素的叶子节点个数合并给当前元素。

最后记得统计每个叶子节点自己的贡献即可。

SCOI 2012 喵星球上的点名

有 n 个的姓名，分为姓和名两个部分，各为一个字符串。
接着有 m 次点名，每次会给出一个点名串，若此串为一个人的姓或名的子串，则这个人要答到。
求每次点名答到的人数，以及每个人答了多少次到。

$n \leq 5 * 10^4, m \leq 10^5$，字符集大小 $\leq 10^4$

既然要判断子串之间的关系，我们可以考虑将姓、名、点名串在内的所有串拼起来，插入分隔符进行后缀排序。同时对每个后缀标明其来自那个人的姓名。

既然要判断子串之间的关系，我们可以考虑将姓、名、点名串在内的所有串拼起来，插入分隔符进行后缀排序。同时对每个后缀标明其来自那个人的姓名。

对于一个完整的点名串，设其长度为 L ，那么答到的人的姓名一定是后缀排序后，在点名串排名后连续的一段长度 $\geq L$ 的串。则我们现在可以得到一个点名串对应的区间是哪段。

对于第一问，就转化成了区间内有多少个不同的人。这就很莫队了。

对于第一问，就转化成了区间内有多少个不同的人。这就很莫队了。

对于第二问，我们在莫队的时候可以这样操作：当新记录一个人的时候，给他的次数加上剩余询问数，当一个人被删除时，减去剩余询问数即可。

NOI 2015 品酒大会

给出长度为 n 的字符串 s，每个字符都有一个值 a_i，若选择了以 i 开头的后缀和以 j 开头的后缀，则价值为 $a_i * a_j$。
求对于每个值 $len \in [0, n)$，有多少对后缀满足它们的公共前缀长度 $\geq len$，同时，求出在这些满足条件的后缀中，价值的最大值。

$n \leq 3 * 10^5$

首先发现，对于一对后缀，若它们的公共前缀长度为 k，那么它们的贡献将同时能够计算到 $len \in [0, k)$ 的答案中。则 len 越小，答案必定越大。

首先发现，对于一对后缀，若它们的公共前缀长度为 k，那么它们的贡献将同时能够计算到 $len \in [0, k)$ 的答案中。则 len 越小，答案必定越大。

则我们后缀排序后，只需要关心公共前缀恰好 $= len$ 的答案，若记 ans_i 为 $len = i$ 时的答案，则我们最后对 ans 取后缀和即可。

那么处理过程就变得单调了，我们考虑下一步怎么做。由于对于一对 len 相似的酒，它一定会在 1 至 $len-1$ 相似的时候被枚举过，那么我们要考虑省去这个枚举 1 到 $len-1$ 相似的答案的过程，否则复杂度还会是 n^2 的。

那么处理过程就变得单调了，我们考虑下一步怎么做。由于对于一对 len 相似的酒，它一定会在 1 至 $len-1$ 相似的时候被枚举过，那么我们要考虑省去这个枚举 1 到 $len-1$ 相似的答案的过程，否则复杂度还会是 n^2 的。

怎么做呢？首先可以发现，我们必须要由大到小来枚举 len ，否则无法省去，那我们不难想到，将后缀数组的 $height$ 数组从大到小排序，然后从大到小处理。

这一步倒是简单，但是注意到，当我们处理到某个长度 $height_i = k$ 的时候，如果 $height_{i+1} = l > k$，我们显然不能直接单纯地计算 $height_i$ 相连的两个后缀 sa_i 和 sa_{i-1}，因为第 sa_i 个后缀与第 sa_{i+1} 个后缀它们也是可以对长度 k 的答案做出贡献的，而且这个贡献无关 $len = l$ 的答案。因为计算 l 答案时我们只关注了 sa_i 与 sa_{i+1} 这一对后缀而已。

怎么办呢？于是我们发现，我们完全可以维护一个并查集，每次处理一个 $height$ 数组就将它所代表的两个后缀放在一个并查集里，处理 $height$ 数组的时候并不单纯处理两个后缀，而是处理两个后缀所在的并查集。

怎么办呢？于是我们发现，我们完全可以维护一个并查集，每次处理一个 $height$ 数组就将它所代表的两个后缀放在一个并查集里，处理 $height$ 数组的时候并不单纯处理两个后缀，而是处理两个后缀所在的并查集。

由于我们是按 $height$ 数组从大到小处理的，所以对于我当前处理的一对重合度为 k 的后缀，它们所在的两个并查集内的后缀必定都是重合度大于等于 k 的，则也都可以计算贡献。

那么题目的解法就出来了：

- 后缀排序并处理出 $height$ 数组。
- 按 $height$ 从大到小的顺序来处理答案，每次将这两个后缀所在的并查集合并，并计算 $height$ 相似的答案。
- 并查集维护 $size$ ，最大值和最小值，之所以要维护最小值，就是因为题目内的权值可以为负数，两个很小的负数乘起来倒还是一个坑点。
- 对于答案的计算，则是每次合并两个并查集的时候，$height$ 相似的数量类答案增加两个 $size$ 的乘积，最大值答案在两个并查集的 max 乘积与 min 乘积中取最大值。
- 统计答案的后缀和与后缀最大值

NOI 2016 优秀的拆分

若一个字符串是优秀的，满足其能够划分成连续的四段，且第一段等于第二段，第三段等于第四段，即可划分成 $AABB$ 的形式。
现给出 T 个字符串，求每个字符串有多少个优秀的子串。

$T \leq 10$，字符串长度 ≤ 30000

先考虑一个暴力，那就是对于每个点 i ，求出以 i 为右端点的，可以划分为两个相同串（也就是形如 AA）的子串个数 a_i，以及以 i 为左端点的可划分为 AA 式的子串个数。那么答案就是 $\sum a_i * b_{i+1}$。这样是 n^2 的，可以获得 95 分的好成绩。(说实话考场上谁会打正解 $==$)

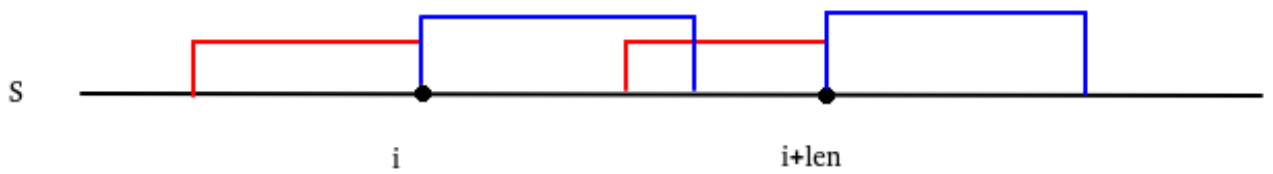
先考虑一个暴力，那就是对于每个点 i ，求出以 i 为右端点的，可以划分为两个相同串（也就是形如 AA）的子串个数 a_i，以及以 i 为左端点的可划分为 AA 式的子串个数。那么答案就是 $\sum a_i * b_{i+1}$。这样是 n^2 的，可以获得 95 分的好成绩。(说实话考场上谁会打正解 $==$)

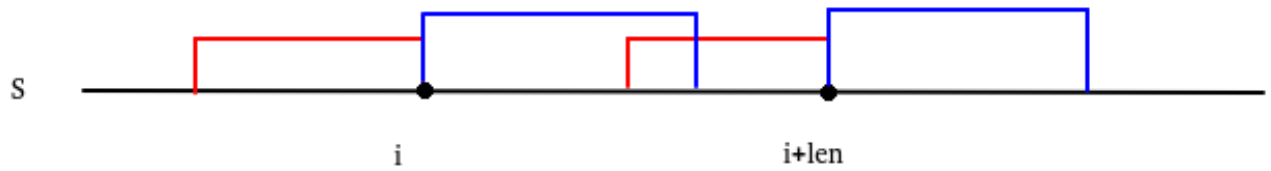
接下来考虑 100 分怎么做。~~我们可以从 darkbzoj.tk 上下载 $.out$ 文件，然后打表输出最后5 分。~~

那 100 分应该就是优化求 a, b 的过程了。考虑对不同长度的 AA 串进行处理。

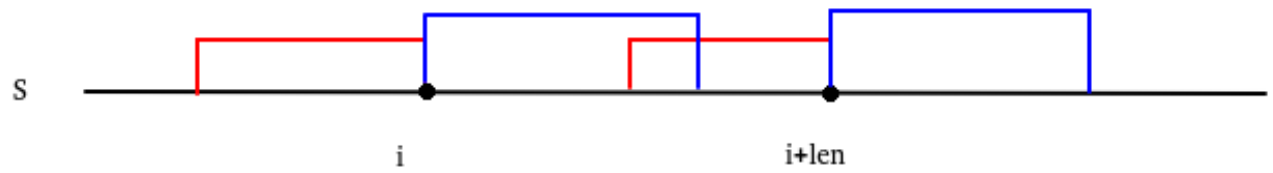
那 100 分应该就是优化求 a, b 的过程了。考虑对不同长度的 AA 串进行处理。

我们可以枚举一个长度 len ，然后每隔 len 标记一个点，若两个相邻的标记点之间的后缀的最长公共前缀 $(LCP)+$ 前缀的最长公共后缀 $(LCS) \geq len$ 的话，就是如下情况：

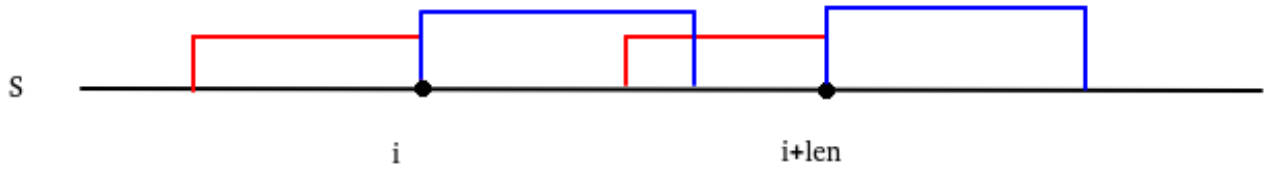




蓝色的即为 LCP ，红色部分为 LCS 。设它们的重合部分长度为 $same$ ，对于这种情况，我们不难发现，对于 $i-LCS$ 到 $i-LCS+same$ 的位置，以它们为开头的长度为 $2*len$ 的串必定为 AA 形式的串，因为从中间分开必定相等，实在想不出来在纸上画一画就行。



那么同理，以 $i-LCS+2*len-1$ 到 $i-LCS+same+2*len-1$ 结尾的长度为 $2*len$ 的串也必定为 AA 形式。



则我们对于相邻的两个距离为 len 的点，如果它们的 $LCP + LCS \geq len$，那么对于 $b[i - LCS]$ 至 $b[i - LCS + same]$ 的位置，我们都加上 1，对于 $a[i - LCS + 2 * len - 1]$ 至 $a[i - LCS + same + 2 * len - 1$ 的位置，我们也都加上一，这便是当前这两个距离为 len 的点对 a, b 数组的贡献了。

对于这个加上去的过程，我们可以使用差分。
那么答案就很明显可以直接统计了。

总结下来我们的步骤如下：

- 枚举 len ，每隔 len 便标记一个点，这个的复杂度用调和级数即可证明，复杂度为 $O(nlog)$ 。
- 差分数组更改，复杂度 $O(1)$
- 统计答案，复杂度 $O(n)$