字符串问题选讲

Claris

Hangzhou Dianzi University

2017 年 8 月 3 日

Overview

- 面对字符串问题，有哪些方法？

Overview

- 面对字符串问题，有哪些方法？
- 字符串 Hash

Overview

- 面对字符串问题，有哪些方法？
- 字符串 Hash
- Trie

Overview

- 面对字符串问题，有哪些方法？
- 字符串 Hash
- Trie
- KMP/AC 自动机

Overview

- 面对字符串问题，有哪些方法？
- 字符串 Hash
- Trie
- KMP/AC 自动机
- 后缀数组/树/自动机/平衡树

Overview

- 面对字符串问题，有哪些方法？
- 字符串 Hash
- Trie
- KMP/AC 自动机
- 后缀数组/树/自动机/平衡树
- 面对回文串有：Manacher/回文树

Overview

- 面对字符串问题，有哪些方法？
- 字符串 Hash
- Trie
- KMP/AC 自动机
- 后缀数组/树/自动机/平衡树
- 面对回文串有：Manacher/回文树
- 最重要的是：分析与思考

DNA

给定两个字符串 S 与 T，求 T 在 S 中的出现次数，允许 3 次失配。

DNA

给定两个字符串 S 与 T，求 T 在 S 中的出现次数，允许 3 次失配。

- $1 \leq |T| \leq |S| \leq 100000$。

DNA

给定两个字符串 S 与 T，求 T 在 S 中的出现次数，允许 3 次失配。

- $1 \leq |T| \leq |S| \leq 100000$。
- Source：TJOI 2017

Solution

- 枚举 T 的第一个字符应该匹配 S 中的哪一个位置，判断是否可行。

Solution

- 枚举 T 的第一个字符应该匹配 S 中的哪一个位置，判断是否可行。
- 判断方法：求 LCP，忽略下一个字符，求 LCP，忽略...

Solution

- 枚举 T 的第一个字符应该匹配 S 中的哪一个位置，判断是否可行。
- 判断方法：求 LCP，忽略下一个字符，求 LCP，忽略...
- 如何求 LCP？

Solution

- 枚举 T 的第一个字符应该匹配 S 中的哪一个位置，判断是否可行。
- 判断方法：求 LCP，忽略下一个字符，求 LCP，忽略...
- 如何求 LCP？
- 二分 LCP 的长度，检查 Hash 值是否相等。

Solution

- 枚举 T 的第一个字符应该匹配 S 中的哪一个位置，判断是否可行。
- 判断方法：求 LCP，忽略下一个字符，求 LCP，忽略...
- 如何求 LCP？
- 二分 LCP 的长度，检查 Hash 值是否相等。
- 时间复杂度 $O(|S| \log |S|)$。

A Horrible Poem

如果字符串 B 是字符串 A 的循环节，那么 A 可以由 B 重复若干次得到。

A Horrible Poem

如果字符串 B 是字符串 A 的循环节，那么 A 可以由 B 重复若干次得到。

给定长度为 n 的字符串 S，q 次询问，每次指定 S 的某个子串，求其最短循环节的长度。

A Horrible Poem

如果字符串 B 是字符串 A 的循环节，那么 A 可以由 B 重复若干次得到。

给定长度为 n 的字符串 S，q 次询问，每次指定 S 的某个子串，求其最短循环节的长度。

- $1 \leq n \leq 500000$。

A Horrible Poem

如果字符串 B 是字符串 A 的循环节，那么 A 可以由 B 重复若干次得到。

给定长度为 n 的字符串 S，q 次询问，每次指定 S 的某个子串，求其最短循环节的长度。

- $1 \leq n \leq 500000$。
- $1 \leq q \leq 2000000$。

A Horrible Poem

如果字符串 B 是字符串 A 的循环节，那么 A 可以由 B 重复若干次得到。

给定长度为 n 的字符串 S，q 次询问，每次指定 S 的某个子串，求其最短循环节的长度。

- $1 \leq n \leq 500000$。
- $1 \leq q \leq 2000000$。
- Source： POI 2012

Solution

- 如何判断长度为 m 的子串的循环节长度是否为 k？

Solution

- 如何判断长度为 m 的子串的循环节长度是否为 k？
- 只需要判断长度为 $m - k$ 的前后缀是否相同即可，利用 Hash 就可以解决这个问题。

Solution

- 如何判断长度为 m 的子串的循环节长度是否为 k？
- 只需要判断长度为 $m - k$ 的前后缀是否相同即可，利用 Hash 就可以解决这个问题。
- 如何求最短循环节长度？

Solution

- 如何判断长度为 m 的子串的循环节长度是否为 k？
- 只需要判断长度为 $m - k$ 的前后缀是否相同即可，利用 Hash 就可以解决这个问题。
- 如何求最短循环节长度？
- 循环节长度一定是 m 的约数。

Solution

- 如何判断长度为 m 的子串的循环节长度是否为 k？
- 只需要判断长度为 $m-k$ 的前后缀是否相同即可，利用 Hash 就可以解决这个问题。
- 如何求最短循环节长度？
- 循环节长度一定是 m 的约数。
- 枚举 m 的质因子 p，判断 $\frac{m}{p}$ 是否是循环节，若是则除以 p。

Solution

- 如何判断长度为 m 的子串的循环节长度是否为 k？
- 只需要判断长度为 $m-k$ 的前后缀是否相同即可，利用 Hash 就可以解决这个问题。
- 如何求最短循环节长度？
- 循环节长度一定是 m 的约数。
- 枚举 m 的质因子 p，判断 $\frac{m}{p}$ 是否是循环节，若是则除以 p。
- 时间复杂度 $O(q \log n)$。

Circle of Stones

给定一个长度为 n 的环形字符串 S。

Circle of Stones

给定一个长度为 n 的环形字符串 S。

对于每个 0 到 $n-1$ 的 k 判断：

Circle of Stones

给定一个长度为 n 的环形字符串 S。

对于每个 0 到 $n-1$ 的 k 判断：

能否从 S 中拿走连续 k 个字符，然后将剩下部分连起来得到一个新的环形字符串 T，满足 T 中相邻两个字符都不同。

Circle of Stones

给定一个长度为 n 的环形字符串 S。

对于每个 0 到 $n-1$ 的 k 判断：

能否从 S 中拿走连续 k 个字符，然后将剩下部分连起来得到一个新的环形字符串 T，满足 T 中相邻两个字符都不同。

- $1 \leq n \leq 1000000$。

Circle of Stones

给定一个长度为 n 的环形字符串 S。

对于每个 0 到 $n-1$ 的 k 判断：

能否从 S 中拿走连续 k 个字符，然后将剩下部分连起来得到一个新的环形字符串 T，满足 T 中相邻两个字符都不同。

- $1 \leq n \leq 1000000$。
- Source：XIII Open Cup named after E.V. Pankratiev. GP of Saratov

Solution

- 首先将环倍长，破环成链，那么剩下的子串要满足相邻字符不同且首尾字符不同。

Solution

- 首先将环倍长，破环成链，那么剩下的子串要满足相邻字符不同且首尾字符不同。
- 通过双指针求出每一段极长的子串，满足相邻字符不同。

Solution

- 首先将环倍长，破环成链，那么剩下的子串要满足相邻字符不同且首尾字符不同。
- 通过双指针求出每一段极长的子串，满足相邻字符不同。
- 设这一段长度为 len，从 2 到 len 枚举剩下的串的长度 L，判断是否可行。

Solution

- 首先将环倍长，破环成链，那么剩下的子串要满足相邻字符不同且首尾字符不同。
- 通过双指针求出每一段极长的子串，满足相邻字符不同。
- 设这一段长度为 len，从 2 到 len 枚举剩下的串的长度 L，判断是否可行。
- 若该子串长度为 $len - L + 1$ 的前后缀不能完全匹配，则说明存在距离为 $L - 1$ 的位置不同，也就能充当首尾。

Solution

- 首先将环倍长，破环成链，那么剩下的子串要满足相邻字符不同且首尾字符不同。
- 通过双指针求出每一段极长的子串，满足相邻字符不同。
- 设这一段长度为 len，从 2 到 len 枚举剩下的串的长度 L，判断是否可行。
- 若该子串长度为 $len - L + 1$ 的前后缀不能完全匹配，则说明存在距离为 $L - 1$ 的位置不同，也就能充当首尾。
- Hash 判断即可。

Solution

- 首先将环倍长，破环成链，那么剩下的子串要满足相邻字符不同且首尾字符不同。
- 通过双指针求出每一段极长的子串，满足相邻字符不同。
- 设这一段长度为 len，从 2 到 len 枚举剩下的串的长度 L，判断是否可行。
- 若该子串长度为 $len - L + 1$ 的前后缀不能完全匹配，则说明存在距离为 $L - 1$ 的位置不同，也就能充当首尾。
- Hash 判断即可。
- 时间复杂度 $O(n)$。

N-dimension Matching

给两个 n 维字符串 S 与 T，求 T 在 S 中出现次数。

N-dimension Matching

给两个 n 维字符串 S 与 T，求 T 在 S 中出现次数。

- $1 \leq n \leq 10$。

N-dimension Matching

给两个 n 维字符串 S 与 T，求 T 在 S 中出现次数。

- $1 \leq n \leq 10$。
- $1 \leq |T| \leq |S| \leq 500000$。

N-dimension Matching

给两个 n 维字符串 S 与 T，求 T 在 S 中出现次数。

- $1 \leq n \leq 10$。
- $1 \leq |T| \leq |S| \leq 500000$。
- Source：POJ 2842

Solution

- 枚举 T 的第一个字符应该匹配 S 中的哪一个位置，判断是否可行。

Solution

- 枚举 T 的第一个字符应该匹配 S 中的哪一个位置，判断是否可行。
- 检查 n 维 Hash 值即可 $O(1)$ 判断是否可行。

Solution

- 枚举 T 的第一个字符应该匹配 S 中的哪一个位置，判断是否可行。
- 检查 n 维 Hash 值即可 $O(1)$ 判断是否可行。
- 如何求出 n 维 Hash 值？

Solution

- 枚举 T 的第一个字符应该匹配 S 中的哪一个位置，判断是否可行。
- 检查 n 维 Hash 值即可 $O(1)$ 判断是否可行。
- 如何求出 n 维 Hash 值？
- 一维一维考虑，类比高维前缀和的求法。

Solution

- 枚举 T 的第一个字符应该匹配 S 中的哪一个位置，判断是否可行。
- 检查 n 维 Hash 值即可 $O(1)$ 判断是否可行。
- 如何求出 n 维 Hash 值？
- 一维一维考虑，类比高维前缀和的求法。
- 时间复杂度 $O(n|S|)$。

Country

有一个超长的只包含小写字母的字符串 S，但是其中有一些重复的子串，我们可以把这些子串压缩成一个大写字母。

Country

有一个超长的只包含小写字母的字符串 S，但是其中有一些重复的子串，我们可以把这些子串压缩成一个大写字母。

比如 A =great，B =AniceA，C =BAxB，则 C =greatnicegreatgreatxgreatnicegreat。

Country

有一个超长的只包含小写字母的字符串 S，但是其中有一些重复的子串，我们可以把这些子串压缩成一个大写字母。

比如 A =great，B =AniceA，C =BAxB，则 C =greatnicegreatgreatxgreatnicegreat。

再给定一个不含缩写的字符串 T，求 T 在 S 中的出现次数，答案模 10000。

Country

有一个超长的只包含小写字母的字符串 S，但是其中有一些重复的子串，我们可以把这些子串压缩成一个大写字母。

比如 A =great，B =AniceA，C =BAxB，则 C =greatnicegreatgreatxgreatnicegreat。

再给定一个不含缩写的字符串 T，求 T 在 S 中的出现次数，答案模 10000。

- $1 \leq |T| \leq 100$。

Country

有一个超长的只包含小写字母的字符串 S，但是其中有一些重复的子串，我们可以把这些子串压缩成一个大写字母。

比如 A =great，B =AniceA，C =BAxB，则 C =greatnicegreatgreatxgreatnicegreat。

再给定一个不含缩写的字符串 T，求 T 在 S 中的出现次数，答案模 10000。

- $1 \leq |T| \leq 100$。
- 大写字母数 ≤ 26，每个串长度不超过 100。

Country

有一个超长的只包含小写字母的字符串 S，但是其中有一些重复的子串，我们可以把这些子串压缩成一个大写字母。

比如 A =great，B =AniceA，C =BAxB，则 C =greatnicegreatgreatxgreatnicegreat。

再给定一个不含缩写的字符串 T，求 T 在 S 中的出现次数，答案模 10000。

- $1 \leq |T| \leq 100$。
- 大写字母数 ≤ 26，每个串长度不超过 100。
- 保证压缩过程不会出现环。

Country

有一个超长的只包含小写字母的字符串 S，但是其中有一些重复的子串，我们可以把这些子串压缩成一个大写字母。

比如 A =great，B =AniceA，C =BAxB，则 C =greatnicegreatgreatxgreatnicegreat。

再给定一个不含缩写的字符串 T，求 T 在 S 中的出现次数，答案模 10000。

- $1 \leq |T| \leq 100$。
- 大写字母数 ≤ 26，每个串长度不超过 100。
- 保证压缩过程不会出现环。
- Source： BZOJ 2061

Solution

- 如果 S 解压后不长，那么 KMP 就可以解决这个问题。

Solution

- 如果 S 解压后不长，那么 KMP 就可以解决这个问题。
- 考虑 KMP 与什么有关。

Solution

- 如果 S 解压后不长，那么 KMP 就可以解决这个问题。
- 考虑 KMP 与什么有关。
- 只关心 KMP 的指针位置。

Solution

- 如果 S 解压后不长，那么 KMP 就可以解决这个问题。
- 考虑 KMP 与什么有关。
- 只关心 KMP 的指针位置。
- 考虑 DP，设 $f_{i,j}$ 表示对于第 i 个压缩串，从 j 开始 KMP，最终 KMP 指针会变成多少。

Solution

- 如果 S 解压后不长，那么 KMP 就可以解决这个问题。
- 考虑 KMP 与什么有关。
- 只关心 KMP 的指针位置。
- 考虑 DP，设 $f_{i,j}$ 表示对于第 i 个压缩串，从 j 开始 KMP，最终 KMP 指针会变成多少。
- 同时求出 $g_{i,j}$ 表示对于第 i 个压缩串，从 j 开始 KMP，中间匹配成功多少次。

Solution

- 如果 S 解压后不长，那么 KMP 就可以解决这个问题。
- 考虑 KMP 与什么有关。
- 只关心 KMP 的指针位置。
- 考虑 DP，设 $f_{i,j}$ 表示对于第 i 个压缩串，从 j 开始 KMP，最终 KMP 指针会变成多少。
- 同时求出 $g_{i,j}$ 表示对于第 i 个压缩串，从 j 开始 KMP，中间匹配成功多少次。
- 时间复杂度 $O(n|T|\sum length)$。

Template

给定一个长度为 n 的字符串 S，从中选取一段长度最短的合法的前缀 p。

Template

给定一个长度为 n 的字符串 S，从中选取一段长度最短的合法的前缀 p。

一个串 p 是合法的，当且仅当将 p 重复填充到 n 个位置后，每个位置被填充恰好一种字符，且最后能形成 S。

Template

给定一个长度为 n 的字符串 S，从中选取一段长度最短的合法的前缀 p。

一个串 p 是合法的，当且仅当将 p 重复填充到 n 个位置后，每个位置被填充恰好一种字符，且最后能形成 S。

如：aabaa 可以填充 aabaaabaabaa。

Template

给定一个长度为 n 的字符串 S，从中选取一段长度最短的合法的前缀 p。

一个串 p 是合法的，当且仅当将 p 重复填充到 n 个位置后，每个位置被填充恰好一种字符，且最后能形成 S。

如：aabaa 可以填充 aabaaabaabaa。

- $1 \leq n \leq 500000$。

Template

给定一个长度为 n 的字符串 S，从中选取一段长度最短的合法的前缀 p。

一个串 p 是合法的，当且仅当将 p 重复填充到 n 个位置后，每个位置被填充恰好一种字符，且最后能形成 S。

如：aabaa 可以填充 aabaaabaabaa。

- $1 \leq n \leq 500000$。
- Source： POI 2005

Solution

- 因为要覆盖最前面和最后面，因此 p 一定既是 S 的前缀，又是 S 的后缀。称这种 p 为 S 的 border。

Solution

- 因为要覆盖最前面和最后面，因此 p 一定既是 S 的前缀，又是 S 的后缀。称这种 p 为 S 的 border。
- 枚举每个 border 作为 p，如何判断是否合法？

Solution

- 因为要覆盖最前面和最后面，因此 p 一定既是 S 的前缀，又是 S 的后缀。称这种 p 为 S 的 border。
- 枚举每个 border 作为 p，如何判断是否合法？
- 求出 p 在 S 中的所有出现位置，那么相邻两个位置的距离差值不能超过 $|p|$。

Solution

- 因为要覆盖最前面和最后面，因此 p 一定既是 S 的前缀，又是 S 的后缀。称这种 p 为 S 的 border。
- 枚举每个 border 作为 p，如何判断是否合法？
- 求出 p 在 S 中的所有出现位置，那么相邻两个位置的距离差值不能超过 $|p|$。
- 根据 KMP 的定义，$next_i$ 表示长度为 i 的前缀的最长 border。

Solution

- 因为要覆盖最前面和最后面，因此 p 一定既是 S 的前缀，又是 S 的后缀。称这种 p 为 S 的 border。
- 枚举每个 border 作为 p，如何判断是否合法？
- 求出 p 在 S 中的所有出现位置，那么相邻两个位置的距离差值不能超过 $|p|$。
- 根据 KMP 的定义，$next_i$ 表示长度为 i 的前缀的最长 border。
- 若将 $next_i$ 视作 i 的父亲，那么会得到一棵树的结构。

Solution

- 因为要覆盖最前面和最后面，因此 p 一定既是 S 的前缀，又是 S 的后缀。称这种 p 为 S 的 border。
- 枚举每个 border 作为 p，如何判断是否合法？
- 求出 p 在 S 中的所有出现位置，那么相邻两个位置的距离差值不能超过 $|p|$。
- 根据 KMP 的定义，$next_i$ 表示长度为 i 的前缀的最长 border。
- 若将 $next_i$ 视作 i 的父亲，那么会得到一棵树的结构。
- 那么 p 的出现位置就是它的整棵子树。

Solution

- 从根向下枚举每个 p，这些 p 会形成一条链。

Solution

- 从根向下枚举每个 p，这些 p 会形成一条链。
- 将整棵树建成链表，当 p 往下走的时候，会对应一些子树的删除，将其从链表中删除。

Solution

- 从根向下枚举每个 p，这些 p 会形成一条链。
- 将整棵树建成链表，当 p 往下走的时候，会对应一些子树的删除，将其从链表中删除。
- 在删除的同时维护链表中相邻两个元素差值的最大值即可。

Solution

- 从根向下枚举每个 p，这些 p 会形成一条链。
- 将整棵树建成链表，当 p 往下走的时候，会对应一些子树的删除，将其从链表中删除。
- 在删除的同时维护链表中相邻两个元素差值的最大值即可。
- 时间复杂度 $O(n)$。

可持久化字符串

一个串 T 是 S 的循环节，当且仅当存在正整数 k，使得 S 是 T^k(即 T 重复 k 次) 的前缀，比如 abcd 是 abcdabcdab 的循环节。

可持久地在线维护字符串 S，支持往末尾添加一个字符 c。

可持久化字符串

一个串 T 是 S 的循环节，当且仅当存在正整数 k，使得 S 是 T^k(即 T 重复 k 次) 的前缀，比如 abcd 是 abcdabcdab 的循环节。

可持久地在线维护字符串 S，支持往末尾添加一个字符 c。

输出每次操作之后的字符串的最短循环节长度。

可持久化字符串

一个串 T 是 S 的循环节，当且仅当存在正整数 k，使得 S 是 T^k(即 T 重复 k 次) 的前缀，比如 abcd 是 abcdabcdab 的循环节。

可持久地在线维护字符串 S，支持往末尾添加一个字符 c。

输出每次操作之后的字符串的最短循环节长度。

- $1 \leq |S| \leq 300000$。

可持久化字符串

一个串 T 是 S 的循环节，当且仅当存在正整数 k，使得 S 是 T^k(即 T 重复 k 次) 的前缀，比如 abcd 是 abcdabcdab 的循环节。

可持久地在线维护字符串 S，支持往末尾添加一个字符 c。

输出每次操作之后的字符串的最短循环节长度。

- $1 \leq |S| \leq 300000$。
- $1 \leq c \leq 300000$。

可持久化字符串

一个串 T 是 S 的循环节，当且仅当存在正整数 k，使得 S 是 T^k(即 T 重复 k 次) 的前缀，比如 abcd 是 abcdabcdab 的循环节。

可持久地在线维护字符串 S，支持往末尾添加一个字符 c。

输出每次操作之后的字符串的最短循环节长度。

- $1 \leq |S| \leq 300000$。
- $1 \leq c \leq 300000$。
- Source：BZOJ 2016 NOI 十连测第五场

Solution

- 考虑 KMP，那么 $ans = length - next(length)$。

Solution

- 考虑 KMP，那么 $ans = length - next(length)$。
- KMP 是均摊线性的，一旦加上可持久之后均摊分析就被破坏了，导致复杂度退化。

Solution

- 考虑 KMP，那么 $ans = length - next(length)$。
- KMP 是均摊线性的，一旦加上可持久之后均摊分析就被破坏了，导致复杂度退化。
- 考虑 KMP 慢在哪里，在于每次失配的时候要按着 $next$ 数组不断往前跳，直到找到某个位置下一个字符可以匹配为止。

Solution

- 考虑 KMP，那么 $ans = length - next(length)$。
- KMP 是均摊线性的，一旦加上可持久之后均摊分析就被破坏了，导致复杂度退化。
- 考虑 KMP 慢在哪里，在于每次失配的时候要按着 $next$ 数组不断往前跳，直到找到某个位置下一个字符可以匹配为止。
- 维护一个转移数组 $trans(x, y)$ 表示如果下一个字符为 y，那么从 x 开始按照 $next$ 跳最终会跳到哪个点。

Solution

- 考虑加入一个字符 c 带来的影响，那么 $next(now) = trans(x, c)$，$trans(now)$ 与 $trans(next(now))$ 是一样的，$trans(x, c)$ 则修改为了 now。

Solution

- 考虑加入一个字符 c 带来的影响，那么 $next(now) = trans(x, c)$，$trans(now)$ 与 $trans(next(now))$ 是一样的，$trans(x, c)$ 则修改为了 now。
- 考虑用可持久化线段树来维护 $trans$，那么复制是 $O(1)$ 的，修改和查询都是 $O(\log c)$ 的。

Solution

- 考虑加入一个字符 c 带来的影响，那么 $next(now) = trans(x, c)$，$trans(now)$ 与 $trans(next(now))$ 是一样的，$trans(x, c)$ 则修改为了 now。
- 考虑用可持久化线段树来维护 $trans$，那么复制是 $O(1)$ 的，修改和查询都是 $O(\log c)$ 的。
- 因为整个字符串本身也要可持久化，所以还需要用另外一棵可持久化线段树来可持久地维护每个点 $trans$ 的线段树的树根编号。

Solution

- 考虑加入一个字符 c 带来的影响，那么 $next(now) = trans(x, c)$，$trans(now)$ 与 $trans(next(now))$ 是一样的，$trans(x, c)$ 则修改为了 now。
- 考虑用可持久化线段树来维护 $trans$，那么复制是 $O(1)$ 的，修改和查询都是 $O(\log c)$ 的。
- 因为整个字符串本身也要可持久化，所以还需要用另外一棵可持久化线段树来可持久地维护每个点 $trans$ 的线段树的树根编号。
- 时间复杂度 $O(|S| \log c)$。

跳蚤

给定一个长度为 n 的字符串 S，请将 S 分成不超过 k 段，使得每一段的所有子串中，字典序最大的子串最小。

跳蚤

给定一个长度为 n 的字符串 S，请将 S 分成不超过 k 段，使得每一段的所有子串中，字典序最大的子串最小。

- $1 \leq k \leq n \leq 100000$。

跳蚤

给定一个长度为 n 的字符串 S，请将 S 分成不超过 k 段，使得每一段的所有子串中，字典序最大的子串最小。

- $1 \leq k \leq n \leq 100000$。
- Source：BZOJ 4310

Solution

- 首先求出后缀数组，得到本质不同的子串的个数。

Solution

- 首先求出后缀数组，得到本质不同的子串的个数。
- 二分答案，每次先通过后缀数组求出第 *mid* 小的子串，然后贪心进行检验。

Solution

- 首先求出后缀数组，得到本质不同的子串的个数。
- 二分答案，每次先通过后缀数组求出第 *mid* 小的子串，然后贪心进行检验。
- 从后往前贪心，每次只会加入一个后缀，且不会对之前加入的串造成影响，如果不能加了，那就划为一段。

Solution

- 首先求出后缀数组，得到本质不同的子串的个数。
- 二分答案，每次先通过后缀数组求出第 mid 小的子串，然后贪心进行检验。
- 从后往前贪心，每次只会加入一个后缀，且不会对之前加入的串造成影响，如果不能加了，那就划为一段。
- 时间复杂度 $O(n \log n)$。

优秀的拆分

如果一个字符串可以被拆分为 $AABB$ 的形式，其中 A 和 B 是任意非空字符串，则我们称该字符串的这种拆分是优秀的。

优秀的拆分

如果一个字符串可以被拆分为 $AABB$ 的形式，其中 A 和 B 是任意非空字符串，则我们称该字符串的这种拆分是优秀的。

例如，对于字符串 aabaabaa，如果令 A =aab，B=a，就是一个优秀的拆分方式。

优秀的拆分

如果一个字符串可以被拆分为 $AABB$ 的形式，其中 A 和 B 是任意非空字符串，则我们称该字符串的这种拆分是优秀的。

例如，对于字符串 aabaabaa，如果令 A =aab，B=a，就是一个优秀的拆分方式。

给定一个字符串 S，求它所有非空子串的所有拆分方式中，优秀拆分的总个数。

优秀的拆分

如果一个字符串可以被拆分为 $AABB$ 的形式，其中 A 和 B 是任意非空字符串，则我们称该字符串的这种拆分是优秀的。

例如，对于字符串 aabaabaa，如果令 A =aab，B=a，就是一个优秀的拆分方式。

给定一个字符串 S，求它所有非空子串的所有拆分方式中，优秀拆分的总个数。

- $1 \leq |S| \leq 300000$。

优秀的拆分

如果一个字符串可以被拆分为 $AABB$ 的形式，其中 A 和 B 是任意非空字符串，则我们称该字符串的这种拆分是优秀的。

例如，对于字符串 aabaabaa，如果令 A =aab，B=a，就是一个优秀的拆分方式。

给定一个字符串 S，求它所有非空子串的所有拆分方式中，优秀拆分的总个数。

- $1 \leq |S| \leq 300000$。
- Source： NOI 2016

Solution

- 如果一个字符串可以写成 uu 的形式，则称它是一个 square。

Solution

- 如果一个字符串可以写成 uu 的形式，则称它是一个 square。
- 设 f_i 表示以 i 结尾的 square 个数，g_i 表示以 i 开头的 square 个数，则 $ans = \sum_{i=1}^{|S|-1} f_i g_{i+1}$。

Solution

- 如果一个字符串可以写成 uu 的形式，则称它是一个 square。
- 设 f_i 表示以 i 结尾的 square 个数，g_i 表示以 i 开头的 square 个数，则 $ans = \sum_{i=1}^{|S|-1} f_i g_{i+1}$。
- 枚举 square 中 u 的长度 L，每 L 步取一个关键点。

Solution

- 如果一个字符串可以写成 uu 的形式，则称它是一个 square。
- 设 f_i 表示以 i 结尾的 square 个数，g_i 表示以 i 开头的 square 个数，则 $ans = \sum_{i=1}^{|S|-1} f_i g_{i+1}$。
- 枚举 square 中 u 的长度 L，每 L 步取一个关键点。
- 那么每个长度为 $2L$ 的 square 的肯定恰好经过两个相邻的关键点，且位置差距为 L 的字符一一匹配。

Solution

- 相邻关键点之间通过后缀数组求出最长公共前后缀，即可知道存在多少该长度的 square。

Solution

- 相邻关键点之间通过后缀数组求出最长公共前后缀，即可知道存在多少该长度的 square。
- 这对应 f 和 g 的一段区间 $+1$，差分前缀和后单点修改即可。

Solution

- 相邻关键点之间通过后缀数组求出最长公共前后缀，即可知道存在多少该长度的 square。
- 这对应 f 和 g 的一段区间 $+1$，差分前缀和后单点修改即可。
- 时间复杂度 $O(|S| \log |S|)$。

Str

给定两个字符串 S 与 T，求最长公共子串的长度，允许一次失配。

Str

给定两个字符串 S 与 T，求最长公共子串的长度，允许一次失配。

- $1 \leq |S| \leq 100000$。

Str

给定两个字符串 S 与 T，求最长公共子串的长度，允许一次失配。

- $1 \leq |S| \leq 100000$。
- $1 \leq |T| \leq 100000$。

Str

给定两个字符串 S 与 T，求最长公共子串的长度，允许一次失配。

- $1 \leq |S| \leq 100000$。
- $1 \leq |T| \leq 100000$。
- Source：BZOJ 3145

Solution

- 我们称失配的位置为模糊点。

Solution

- 我们称失配的位置为模糊点。
- 如果不存在模糊点，答案就是两个串的最长公共子串，经典问题。

Solution

- 我们称失配的位置为模糊点。
- 如果不存在模糊点，答案就是两个串的最长公共子串，经典问题。
- 如果模糊点是某个串的开头或者结尾，可以暴力枚举另一个串中的某个前后缀更新答案。

Solution

- 我们称失配的位置为模糊点。
- 如果不存在模糊点，答案就是两个串的最长公共子串，经典问题。
- 如果模糊点是某个串的开头或者结尾，可以暴力枚举另一个串中的某个前后缀更新答案。
- 否则，假设模糊点在第一个串里是 i，在第二个串里是 j，那么此时对答案的贡献为 $lcp(i+1,j+1)+lcs(i-1,j-1)+1$。

Solution

- 将两个串用特殊字符拼接，求出正串的后缀数组以及反串的后缀数组，那么 $lcp(i+1, j+1) + lcs(i-1, j-1) + 1$ 等价于两个 height 数组的区间最小值的和。

Solution

- 将两个串用特殊字符拼接，求出正串的后缀数组以及反串的后缀数组，那么 $lcp(i+1, j+1) + lcs(i-1, j-1) + 1$ 等价于两个 height 数组的区间最小值的和。
- 考虑从大到小枚举第一个数，每次把所有大于等于它的部分全部合并，那么在另一个数组里显然只有相邻的后缀有用。

Solution

- 将两个串用特殊字符拼接，求出正串的后缀数组以及反串的后缀数组，那么 $lcp(i+1, j+1) + lcs(i-1, j-1) + 1$ 等价于两个 height 数组的区间最小值的和。
- 考虑从大到小枚举第一个数，每次把所有大于等于它的部分全部合并，那么在另一个数组里显然只有相邻的后缀有用。
- 对于每个集合建立两棵平衡树，维护两个串在反串里的 *rank*。

Solution

- 将两个串用特殊字符拼接，求出正串的后缀数组以及反串的后缀数组，那么 $lcp(i+1, j+1) + lcs(i-1, j-1) + 1$ 等价于两个 height 数组的区间最小值的和。
- 考虑从大到小枚举第一个数，每次把所有大于等于它的部分全部合并，那么在另一个数组里显然只有相邻的后缀有用。
- 对于每个集合建立两棵平衡树，维护两个串在反串里的 *rank*。
- 在启发式合并的时候，只需要在另一个串对应的平衡树里找到前驱后继然后更新答案即可。

Solution

- 将两个串用特殊字符拼接，求出正串的后缀数组以及反串的后缀数组，那么 $lcp(i+1, j+1) + lcs(i-1, j-1) + 1$ 等价于两个 height 数组的区间最小值的和。
- 考虑从大到小枚举第一个数，每次把所有大于等于它的部分全部合并，那么在另一个数组里显然只有相邻的后缀有用。
- 对于每个集合建立两棵平衡树，维护两个串在反串里的 *rank*。
- 在启发式合并的时候，只需要在另一个串对应的平衡树里找到前驱后继然后更新答案即可。
- 时间复杂度 $O(n \log^2 n)$。

回忆树

给定一棵 n 个点的无根树，每条边上有一个小写字符。

回忆树

给定一棵 n 个点的无根树，每条边上有一个小写字符。

m 次询问，每次给定一个字符串 T 以及两个点 u, v，从 u 开始沿着最短路线走到 v，将途径的边上的字符按顺序写下来得到串 S。

回忆树

给定一棵 n 个点的无根树，每条边上有一个小写字符。

m 次询问，每次给定一个字符串 T 以及两个点 u, v，从 u 开始沿着最短路线走到 v，将途径的边上的字符按顺序写下来得到串 S。

求 S 中 T 出现的次数。

回忆树

给定一棵 n 个点的无根树，每条边上有一个小写字符。

m 次询问，每次给定一个字符串 T 以及两个点 u, v，从 u 开始沿着最短路线走到 v，将途径的边上的字符按顺序写下来得到串 S。

求 S 中 T 出现的次数。

- $2 \leq n \leq 100000$。

回忆树

给定一棵 n 个点的无根树，每条边上有一个小写字符。

m 次询问，每次给定一个字符串 T 以及两个点 u, v，从 u 开始沿着最短路线走到 v，将途径的边上的字符按顺序写下来得到串 S。

求 S 中 T 出现的次数。

- $2 \leq n \leq 100000$。
- $1 \leq m \leq 100000$。

回忆树

给定一棵 n 个点的无根树，每条边上有一个小写字符。

m 次询问，每次给定一个字符串 T 以及两个点 u, v，从 u 开始沿着最短路线走到 v，将途径的边上的字符按顺序写下来得到串 S。

求 S 中 T 出现的次数。

- $2 \leq n \leq 100000$。
- $1 \leq m \leq 100000$。
- $1 \leq \sum |T| \leq 300000$。

回忆树

给定一棵 n 个点的无根树，每条边上有一个小写字符。

m 次询问，每次给定一个字符串 T 以及两个点 u, v，从 u 开始沿着最短路线走到 v，将途径的边上的字符按顺序写下来得到串 S。

求 S 中 T 出现的次数。

- $2 \leq n \leq 100000$。
- $1 \leq m \leq 100000$。
- $1 \leq \sum |T| \leq 300000$。
- Source：BZOJ 4231

Solution

- 一个长度为 $|T|$ 的串在树上匹配有两种情况：

Solution

- 一个长度为 $|T|$ 的串在树上匹配有两种情况：
- (1) 在 LCA 处转弯，那么这种情况只有 $O(|T|)$ 次，暴力提取出长度为 $2|T|$ 的链进行 KMP 即可。

Solution

- 一个长度为 $|T|$ 的串在树上匹配有两种情况：
- (1) 在 LCA 处转弯，那么这种情况只有 $O(|T|)$ 次，暴力提取出长度为 $2|T|$ 的链进行 KMP 即可。
- (2) 不转弯，那么可以拆成两个点分别往上到某个祖先的询问。

Solution

- 一个长度为 $|T|$ 的串在树上匹配有两种情况：
- (1) 在 LCA 处转弯，那么这种情况只有 $O(|T|)$ 次，暴力提取出长度为 $2|T|$ 的链进行 KMP 即可。
- (2) 不转弯，那么可以拆成两个点分别往上到某个祖先的询问。
- 根据容斥可以进一步将每个询问拆成两个点到根的答案之差。

Solution

- 对所有询问串的正反串建立 AC 自动机，求出 fail 树上每个点的 DFS 序。

Solution

- 对所有询问串的正反串建立 AC 自动机，求出 fail 树上每个点的 DFS 序。
- DFS 原树，记录在 AC 自动机上走到了哪个点，在那个点 $+1$，回溯的时候 -1。

Solution

- 对所有询问串的正反串建立 AC 自动机，求出 fail 树上每个点的 DFS 序。
- DFS 原树，记录在 AC 自动机上走到了哪个点，在那个点 $+1$，回溯的时候 -1。
- 那么一个询问的答案就是 fail 树上的子树和，树状数组维护即可。

Solution

- 对所有询问串的正反串建立 AC 自动机，求出 fail 树上每个点的 DFS 序。
- DFS 原树，记录在 AC 自动机上走到了哪个点，在那个点 $+1$，回溯的时候 -1。
- 那么一个询问的答案就是 fail 树上的子树和，树状数组维护即可。
- 时间复杂度 $O((n+m)\log n+\sum|T|)$。

Crash 和陶陶的游戏

给定两棵有根树 A 和 B，每条树边上有个小写字符。

Crash 和陶陶的游戏

给定两棵有根树 A 和 B，每条树边上有个小写字符。

定义 $A_{s\to t}$ 表示从 s 开始沿着最短路线走到 t，将途径的边上的字符按顺序写下来得到的串。

Crash 和陶陶的游戏

给定两棵有根树 A 和 B，每条树边上有个小写字符。

定义 $A_{s\to t}$ 表示从 s 开始沿着最短路线走到 t，将途径的边上的字符按顺序写下来得到的串。

一开始 A 和 B 都只有根节点 1，有 n 次操作，每次操作会选择其中一棵树，在某个点下面添加一个叶子。

Crash 和陶陶的游戏

给定两棵有根树 A 和 B，每条树边上有个小写字符。

定义 $A_{s\to t}$ 表示从 s 开始沿着最短路线走到 t，将途径的边上的字符按顺序写下来得到的串。

一开始 A 和 B 都只有根节点 1，有 n 次操作，每次操作会选择其中一棵树，在某个点下面添加一个叶子。

请在每次操作之后统计满足条件的三元组 (i, j, k) 个数：

Crash 和陶陶的游戏

给定两棵有根树 A 和 B，每条树边上有个小写字符。

定义 $A_{s\to t}$ 表示从 s 开始沿着最短路线走到 t，将途径的边上的字符按顺序写下来得到的串。

一开始 A 和 B 都只有根节点 1，有 n 次操作，每次操作会选择其中一棵树，在某个点下面添加一个叶子。

请在每次操作之后统计满足条件的三元组 (i,j,k) 个数：

(1)$1 \le i \le |A|, 1 \le j,k \le |B|$，且 j 是 k 的祖先。

Crash 和陶陶的游戏

给定两棵有根树 A 和 B，每条树边上有个小写字符。

定义 $A_{s\to t}$ 表示从 s 开始沿着最短路线走到 t，将途径的边上的字符按顺序写下来得到的串。

一开始 A 和 B 都只有根节点 1，有 n 次操作，每次操作会选择其中一棵树，在某个点下面添加一个叶子。

请在每次操作之后统计满足条件的三元组 (i,j,k) 个数：

(1)$1\le i\le |A|, 1\le j,k\le |B|$，且 j 是 k 的祖先。

(2)$A_{1\to i}=B_{k\to j}$。

Crash 和陶陶的游戏

给定两棵有根树 A 和 B，每条树边上有个小写字符。

定义 $A_{s\to t}$ 表示从 s 开始沿着最短路线走到 t，将途径的边上的字符按顺序写下来得到的串。

一开始 A 和 B 都只有根节点 1，有 n 次操作，每次操作会选择其中一棵树，在某个点下面添加一个叶子。

请在每次操作之后统计满足条件的三元组 (i,j,k) 个数：

(1)$1 \le i \le |A|, 1 \le j,k \le |B|$，且 j 是 k 的祖先。

(2)$A_{1\to i} = B_{k\to j}$。

- $1 \le n \le 100000$。

Crash 和陶陶的游戏

给定两棵有根树 A 和 B，每条树边上有个小写字符。

定义 $A_{s\to t}$ 表示从 s 开始沿着最短路线走到 t，将途径的边上的字符按顺序写下来得到的串。

一开始 A 和 B 都只有根节点 1，有 n 次操作，每次操作会选择其中一棵树，在某个点下面添加一个叶子。

请在每次操作之后统计满足条件的三元组 (i,j,k) 个数：

(1)$1 \le i \le |A|, 1 \le j,k \le |B|$，且 j 是 k 的祖先。

(2)$A_{1\to i} = B_{k\to j}$。

- $1 \le n \le 100000$。
- Source：BZOJ 3467

Solution

- 离线建出两棵树，对 B 中每个点预处理出其往上 2^k 步的父亲以及中间的串的 Hash 值。

Solution

- 离线建出两棵树，对 B 中每个点预处理出其往上 2^k 步的父亲以及中间的串的 Hash 值。
- 通过这个可以在 $O(\log n)$ 的时间内比较两个串的字典序，以此将它们按字典序排序。

Solution

- 离线建出两棵树，对 B 中每个点预处理出其往上 2^k 步的父亲以及中间的串的 Hash 值。
- 通过这个可以在 $O(\log n)$ 的时间内比较两个串的字典序，以此将它们按字典序排序。
- 将树 A 压缩成 Trie，并求出根到每个点的 Hash 值，同时预处理出每个串在 B 数组中出现的区间。

Solution

- 离线建出两棵树，对 B 中每个点预处理出其往上 2^k 步的父亲以及中间的串的 Hash 值。
- 通过这个可以在 $O(\log n)$ 的时间内比较两个串的字典序，以此将它们按字典序排序。
- 将树 A 压缩成 Trie，并求出根到每个点的 Hash 值，同时预处理出每个串在 B 数组中出现的区间。
- 具体的求法是：x 的区间一定是 x 的父亲的区间的子区间，在内部二分找到区间的左右端点即可。

Solution

- 如果是往树 A 中添加节点 x，那么对答案的贡献是 B 数组中对应区间内已加入的点数。

Solution

- 如果是往树 A 中添加节点 x，那么对答案的贡献是 B 数组中对应区间内已加入的点数。
- 如果是往树 B 中添加节点 x，那么可以倍增枚举在 Trie 中可以走多深。

Solution

- 如果是往树 A 中添加节点 x，那么对答案的贡献是 B 数组中对应区间内已加入的点数。
- 如果是往树 B 中添加节点 x，那么可以倍增枚举在 Trie 中可以走多深。
- 根据 Hash 值可以很方便地进行定位，对答案的贡献是某个点到根节点路径上已经加入的点数。

Solution

- 如果是往树 A 中添加节点 x，那么对答案的贡献是 B 数组中对应区间内已加入的点数。
- 如果是往树 B 中添加节点 x，那么可以倍增枚举在 Trie 中可以走多深。
- 根据 Hash 值可以很方便地进行定位，对答案的贡献是某个点到根节点路径上已经加入的点数。
- 用两棵树状数组分别维护即可。

Solution

- 如果是往树 A 中添加节点 x，那么对答案的贡献是 B 数组中对应区间内已加入的点数。
- 如果是往树 B 中添加节点 x，那么可以倍增枚举在 Trie 中可以走多深。
- 根据 Hash 值可以很方便地进行定位，对答案的贡献是某个点到根节点路径上已经加入的点数。
- 用两棵树状数组分别维护即可。
- 时间复杂度 $O(n \log^2 n)$。

Pattern String

对于超长字符串，可以将其压缩成 $(S_1)k_1(S_2)k_2...(S_n)k_n$ 的形式，其中 $(S)k$ 表示 S 重复 k 次。不允许嵌套压缩。

Pattern String

对于超长字符串，可以将其压缩成 $(S_1)k_1(S_2)k_2...(S_n)k_n$ 的形式，其中 $(S)k$ 表示 S 重复 k 次。不允许嵌套压缩。

给定两个解压后等长的压缩串 S 和 T，请判断 S 和 T 是否循环同构。

Pattern String

对于超长字符串，可以将其压缩成 $(S_1)k_1(S_2)k_2...(S_n)k_n$ 的形式，其中 $(S)k$ 表示 S 重复 k 次。不允许嵌套压缩。

给定两个解压后等长的压缩串 S 和 T，请判断 S 和 T 是否循环同构。

- 压缩串长 ≤ 11000。

Pattern String

对于超长字符串，可以将其压缩成 $(S_1)k_1(S_2)k_2...(S_n)k_n$ 的形式，其中 $(S)k$ 表示 S 重复 k 次。不允许嵌套压缩。

给定两个解压后等长的压缩串 S 和 T，请判断 S 和 T 是否循环同构。

- 压缩串长 ≤ 11000。
- 解压后串长 $\leq 2 \cdot 10^8$。

Pattern String

对于超长字符串，可以将其压缩成 $(S_1)k_1(S_2)k_2...(S_n)k_n$ 的形式，其中 $(S)k$ 表示 S 重复 k 次。不允许嵌套压缩。

给定两个解压后等长的压缩串 S 和 T，请判断 S 和 T 是否循环同构。

- 压缩串长 ≤ 11000。
- 解压后串长 $\leq 2 \cdot 10^8$。
- Source： HDU 5509

Solution

- 只要求出两个字符串的最小表示，即可判断是否循环同构。

Solution

- 只要求出两个字符串的最小表示，即可判断是否循环同构。
- 枚举最小表示的开头在哪个位置，然后求出 Hash 值。

Solution

- 只要求出两个字符串的最小表示，即可判断是否循环同构。
- 枚举最小表示的开头在哪个位置，然后求出 Hash 值。
- 如果两个串的 Hash 值集合有交，说明循环同构。

Solution

- 只要求出两个字符串的最小表示，即可判断是否循环同构。
- 枚举最小表示的开头在哪个位置，然后求出 Hash 值。
- 如果两个串的 Hash 值集合有交，说明循环同构。
- 因为串经过压缩，原串的长度很大，不能直接枚举开头。

Solution

- 考虑当开头在某个串 A^k 里某个位置时的性质：

Solution

- 考虑当开头在某个串 A^k 里某个位置时的性质：
- 假设 A^k 全在开头，现在考虑挪动一个 A 到结尾。

Solution

- 考虑当开头在某个串 A^k 里某个位置时的性质：
- 假设 A^k 全在开头，现在考虑挪动一个 A 到结尾。
- 如果挪动之后字典序更小了，那么再挪动一个 A 到结尾，比较条件不变。

Solution

- 考虑当开头在某个串 A^k 里某个位置时的性质：
- 假设 A^k 全在开头，现在考虑挪动一个 A 到结尾。
- 如果挪动之后字典序更小了，那么再挪动一个 A 到结尾，比较条件不变。
- 因此一旦挪动一个 A 之后字典序变小，那么最小表示一定是将 $k-1$ 个 A 全部挪到结尾。

Solution

- 考虑当开头在某个串 A^k 里某个位置时的性质：
- 假设 A^k 全在开头，现在考虑挪动一个 A 到结尾。
- 如果挪动之后字典序更小了，那么再挪动一个 A 到结尾，比较条件不变。
- 因此一旦挪动一个 A 之后字典序变小，那么最小表示一定是将 $k-1$ 个 A 全部挪到结尾。
- 所以对于一个压缩串 A^k，只需要在其第一次重复和最后一次重复的部分枚举开头即可。

Solution

- 考虑当开头在某个串 A^k 里某个位置时的性质：
- 假设 A^k 全在开头，现在考虑挪动一个 A 到结尾。
- 如果挪动之后字典序更小了，那么再挪动一个 A 到结尾，比较条件不变。
- 因此一旦挪动一个 A 之后字典序变小，那么最小表示一定是将 $k-1$ 个 A 全部挪到结尾。
- 所以对于一个压缩串 A^k，只需要在其第一次重复和最后一次重复的部分枚举开头即可。
- 枚举总长度不超过 $2|S|$。

Solution

- 还有一个问题：如何求 Hash 值？

Solution

- 还有一个问题：如何求 Hash 值？
- 对于开头与结尾，Hash 值可以直接递推计算。

Solution

- 还有一个问题：如何求 Hash 值？
- 对于开头与结尾，Hash 值可以直接递推计算。
- 对于中间 $k-2$ 次移动，可以用矩阵快速幂计算。

Solution

- 还有一个问题：如何求 Hash 值？
- 对于开头与结尾，Hash 值可以直接递推计算。
- 对于中间 $k-2$ 次移动，可以用矩阵快速幂计算。
- 时间复杂度 $O(|S| \log |S|)$。

Prefixuffix

给定一个长度为 n 的字符串 S，求最大的满足条件的 L：

Prefixuffix

给定一个长度为 n 的字符串 S，求最大的满足条件的 L：

$(1)2L \leq n$。

Prefixuffix

给定一个长度为 n 的字符串 S，求最大的满足条件的 L：

(1)$2L \leq n$。

(2)S 的长度为 L 的前后缀循环同构。

Prefixuffix

给定一个长度为 n 的字符串 S，求最大的满足条件的 L：

(1)$2L \leq n$。

(2)S 的长度为 L 的前后缀循环同构。

- $1 \leq n \leq 1000000$。

Prefixuffix

给定一个长度为 n 的字符串 S，求最大的满足条件的 L：

(1)$2L \leq n$。

(2)S 的长度为 L 的前后缀循环同构。

- $1 \leq n \leq 1000000$。
- Source： POI 2012

Solution

- 枚举所有 L，判断是否合法。

Solution

- 枚举所有 L，判断是否合法。
- 如果 x 和 y 循环同构，那么 $x = uv, y = vu$。

Solution

- 枚举所有 L，判断是否合法。
- 如果 x 和 y 循环同构，那么 $x = uv, y = vu$。
- 考虑两个长度都为 n 的字符串 $x_{1..n}$ 和 $y_{1..n}$，定义字符串 $C(x, y) = x_1 y_n x_2 y_{n-1} \ldots x_{n-1} y_2 x_n y_1$。

Solution

- 枚举所有 L，判断是否合法。
- 如果 x 和 y 循环同构，那么 $x = uv, y = vu$。
- 考虑两个长度都为 n 的字符串 $x_{1..n}$ 和 $y_{1..n}$，定义字符串 $C(x, y) = x_1 y_n x_2 y_{n-1} \ldots x_{n-1} y_2 x_n y_1$。
- 例如：$x =$abcd，$y =$bcda，则 $C(x, y) =$aabdccdb。

Solution

- 枚举所有 L，判断是否合法。
- 如果 x 和 y 循环同构，那么 $x = uv, y = vu$。
- 考虑两个长度都为 n 的字符串 $x_{1..n}$ 和 $y_{1..n}$，定义字符串 $C(x, y) = x_1 y_n x_2 y_{n-1} \cdots x_{n-1} y_2 x_n y_1$。
- 例如：$x =$abcd，$y =$bcda，则 $C(x, y) =$aabdccdb。
- $C(x, x)$ 显然是一个长度为偶数的回文串。

Solution

- 枚举所有 L，判断是否合法。
- 如果 x 和 y 循环同构，那么 $x = uv, y = vu$。
- 考虑两个长度都为 n 的字符串 $x_{1..n}$ 和 $y_{1..n}$，定义字符串 $C(x, y) = x_1 y_n x_2 y_{n-1} \cdots x_{n-1} y_2 x_n y_1$。
- 例如：$x =$abcd，$y =$bcda，则 $C(x, y) =$aabdccdb。
- $C(x, x)$ 显然是一个长度为偶数的回文串。
- 令 $x = uv, y = vu$，则 $C(x, y) = C(u, u) + C(v, v)$，那么这是由两个偶回文串拼接而成的偶回文串。

Solution

- 枚举所有 L，判断是否合法。
- 如果 x 和 y 循环同构，那么 $x = uv, y = vu$。
- 考虑两个长度都为 n 的字符串 $x_{1..n}$ 和 $y_{1..n}$，定义字符串 $C(x, y) = x_1 y_n x_2 y_{n-1} \cdots x_{n-1} y_2 x_n y_1$。
- 例如：x =abcd，y =bcda，则 $C(x, y)$ =aabdccdb。
- $C(x, x)$ 显然是一个长度为偶数的回文串。
- 令 $x = uv, y = vu$，则 $C(x, y) = C(u, u) + C(v, v)$，那么这是由两个偶回文串拼接而成的偶回文串。
- 问题转化为判断一个偶回文串是否可以由两个偶回文串拼接而成。

Solution

- 引理：若 $s = x_1x_2 = y_1y_2 = z_1z_2, |x_1| < |y_1| < |z_1|$，其中 x_2, y_1, y_2, z_1 都是回文串，那么 x_1 和 z_2 也是回文串。

Solution

- 引理：若 $s = x_1x_2 = y_1y_2 = z_1z_2, |x_1| < |y_1| < |z_1|$，其中 x_2, y_1, y_2, z_1 都是回文串，那么 x_1 和 z_2 也是回文串。
- 令 $s = aa^Rbb^R$，即 s 由两个偶回文串拼接而成，那么根据上面的引理，要么 aa^R 是 s 的最长偶回文前缀，要么 bb^R 是 s 的最长偶回文后缀。

Solution

- 引理：若 $s = x_1x_2 = y_1y_2 = z_1z_2, |x_1| < |y_1| < |z_1|$，其中 x_2, y_1, y_2, z_1 都是回文串，那么 x_1 和 z_2 也是回文串。
- 令 $s = aa^Rbb^R$，即 s 由两个偶回文串拼接而成，那么根据上面的引理，要么 aa^R 是 s 的最长偶回文前缀，要么 bb^R 是 s 的最长偶回文后缀。
- 将字符串 S 沿着中点按上述方法对折，得到字符串 T。

Solution

- 引理：若 $s = x_1x_2 = y_1y_2 = z_1z_2, |x_1| < |y_1| < |z_1|$，其中 x_2, y_1, y_2, z_1 都是回文串，那么 x_1 和 z_2 也是回文串。
- 令 $s = aa^Rbb^R$，即 s 由两个偶回文串拼接而成，那么根据上面的引理，要么 aa^R 是 s 的最长偶回文前缀，要么 bb^R 是 s 的最长偶回文后缀。
- 将字符串 S 沿着中点按上述方法对折，得到字符串 T。
- 那么若 L 合法，则 T 中长度为 $2L$ 的前缀由两个偶回文串拼接而成。

Solution

- 引理：若 $s = x_1x_2 = y_1y_2 = z_1z_2, |x_1| < |y_1| < |z_1|$，其中 x_2, y_1, y_2, z_1 都是回文串，那么 x_1 和 z_2 也是回文串。
- 令 $s = aa^Rbb^R$，即 s 由两个偶回文串拼接而成，那么根据上面的引理，要么 aa^R 是 s 的最长偶回文前缀，要么 bb^R 是 s 的最长偶回文后缀。
- 将字符串 S 沿着中点按上述方法对折，得到字符串 T。
- 那么若 L 合法，则 T 中长度为 $2L$ 的前缀由两个偶回文串拼接而成。
- 只需要求出每个前缀的最长偶回文前缀以及最长偶回文后缀即可。

Solution

- 利用 Manacher 算法可以得到以每个位置为中心的最长回文半径。

Solution

- 利用 Manacher 算法可以得到以每个位置为中心的最长回文半径。
- 利用这个数组可以很简单得求出每个前缀的最长回文前缀以及判定一个子串是否回文。

Solution

- 利用 Manacher 算法可以得到以每个位置为中心的最长回文半径。
- 利用这个数组可以很简单得求出每个前缀的最长回文前缀以及判定一个子串是否回文。
- 一个前缀的最长偶回文后缀，等价于寻找覆盖了它的离它最远的回文中心。

Solution

- 利用 Manacher 算法可以得到以每个位置为中心的最长回文半径。
- 利用这个数组可以很简单得求出每个前缀的最长回文前缀以及判定一个子串是否回文。
- 一个前缀的最长偶回文后缀，等价于寻找覆盖了它的离它最远的回文中心。
- 进行一次递推即可求出每个前缀的最长偶回文前缀。

Solution

- 利用 Manacher 算法可以得到以每个位置为中心的最长回文半径。
- 利用这个数组可以很简单得求出每个前缀的最长回文前缀以及判定一个子串是否回文。
- 一个前缀的最长偶回文后缀，等价于寻找覆盖了它的离它最远的回文中心。
- 进行一次递推即可求出每个前缀的最长偶回文前缀。
- 时间复杂度 $O(n)$。

课后习题

- Borderless Words (ASC 47)

课后习题

- Borderless Words (ASC 47)
- [Scoi2016] 围棋 (BZOJ 4572)

课后习题

- Borderless Words (ASC 47)
- [Scoi2016] 围棋 (BZOJ 4572)
- GRE Words (HDU 4117)

课后习题

- Borderless Words (ASC 47)
- [Scoi2016] 围棋 (BZOJ 4572)
- GRE Words (HDU 4117)
- Trie (HDU 5081)

课后习题

- Borderless Words (ASC 47)
- [Scoi2016] 围棋 (BZOJ 4572)
- GRE Words (HDU 4117)
- Trie (HDU 5081)
- [Sdoi2016] 生成魔咒 (BZOJ 4516)

课后习题

- Borderless Words (ASC 47)
- [Scoi2016] 围棋 (BZOJ 4572)
- GRE Words (HDU 4117)
- Trie (HDU 5081)
- [Sdoi2016] 生成魔咒 (BZOJ 4516)
- [Tjoi2016&Heoi2016] 字符串 (BZOJ 4556)

课后习题

- Borderless Words (ASC 47)
- [Scoi2016] 围棋 (BZOJ 4572)
- GRE Words (HDU 4117)
- Trie (HDU 5081)
- [Sdoi2016] 生成魔咒 (BZOJ 4516)
- [Tjoi2016&Heoi2016] 字符串 (BZOJ 4556)
- 品酒大会 (NOI 2015)

课后习题

- Borderless Words (ASC 47)
- [Scoi2016] 围棋 (BZOJ 4572)
- GRE Words (HDU 4117)
- Trie (HDU 5081)
- [Sdoi2016] 生成魔咒 (BZOJ 4516)
- [Tjoi2016&Heoi2016] 字符串 (BZOJ 4556)
- 品酒大会 (NOI 2015)
- Cycle (HDU 5782)

Thank you!