文本校正 (fix) 解题报告

Claris

2017 年 4 月 27 日

文本校正 (fix.c/cpp/pas)

给定两个长度都为 n 的字符串 S 和 T，要求将 T 分成 3 个非空连续子串并按任意顺序拼接得到 S。

文本校正 (fix.c/cpp/pas)

给定两个长度都为 n 的字符串 S 和 T，要求将 T 分成 3 个非空连续子串并按任意顺序拼接得到 S。

- 对于 10% 的数据，$n \leq 6$。

文本校正 (fix.c/cpp/pas)

给定两个长度都为 n 的字符串 S 和 T，要求将 T 分成 3 个非空连续子串并按任意顺序拼接得到 S。

- 对于 10% 的数据，$n \leq 6$。
- 对于 30% 的数据，$n \leq 2000$。

文本校正 (fix.c/cpp/pas)

给定两个长度都为 n 的字符串 S 和 T，要求将 T 分成 3 个非空连续子串并按任意顺序拼接得到 S。

- 对于 10% 的数据，$n \leq 6$。
- 对于 30% 的数据，$n \leq 2000$。
- 对于 60% 的数据，$n \leq 200000$。

文本校正 (fix.c/cpp/pas)

给定两个长度都为 n 的字符串 S 和 T，要求将 T 分成 3 个非空连续子串并按任意顺序拼接得到 S。

- 对于 10% 的数据，$n \leq 6$。
- 对于 30% 的数据，$n \leq 2000$。
- 对于 60% 的数据，$n \leq 200000$。
- 对于 100% 的数据，$n \leq 1000000$。

10 分做法

- 对于 10% 的数据，$n \leq 6$。

10 分做法

- 对于 10% 的数据，$n \leq 6$。
- 枚举断点以及拼接顺序然后暴力判断是否合法即可。

10 分做法

- 对于 10% 的数据，$n \leq 6$。
- 枚举断点以及拼接顺序然后暴力判断是否合法即可。
- 时间复杂度 $O(n^3)$。

30 分做法

- 对于 30% 的数据，$n \leq 2000$。

30 分做法

- 对于 30% 的数据，$n \leq 2000$。
- 10 分做法的瓶颈在于判断两个子串是否相等。

30 分做法

- 对于 30% 的数据，$n \leq 2000$。
- 10 分做法的瓶颈在于判断两个子串是否相等。
- 利用 Hash 就可以解决这个问题。

30 分做法

- 对于 30% 的数据，$n \leq 2000$。
- 10 分做法的瓶颈在于判断两个子串是否相等。
- 利用 Hash 就可以解决这个问题。
- 时间复杂度 $O(n^2)$。

60 分做法

- 对于 60% 的数据，$n \leq 200000$。

60 分做法

- 对于 60% 的数据，$n \leq 200000$。
- 不妨假设可以分割成不多于 3 段，那么在输出的时候强行分成 3 段即可。

60 分做法

- 对于 60% 的数据，$n \leq 200000$。
- 不妨假设可以分割成不多于 3 段，那么在输出的时候强行分成 3 段即可。
- 令 $S = ABC$，那么一共有 3 种可能的拼接方案。

60 分做法

- 1.$S = ABC$, $T = BAC$。

60 分做法

- 1.$S = ABC$, $T = BAC$。
- 枚举 T 的每一个前缀 i，判断 $S[i+1, n]$ 是否等于 $T[i+1, n]$ 以及 $S[1, i]$ 与 $T[1, i]$ 是否循环同构。

60 分做法

- 1.$S = ABC$, $T = BAC$。
- 枚举 T 的每一个前缀 i，判断 $S[i+1, n]$ 是否等于 $T[i+1, n]$ 以及 $S[1, i]$ 与 $T[1, i]$ 是否循环同构。
- 判断相等只需要比较 Hash 值是否相等。

60 分做法

- 1.$S = ABC, T = BAC$。
- 枚举 T 的每一个前缀 i，判断 $S[i+1, n]$ 是否等于 $T[i+1, n]$ 以及 $S[1, i]$ 与 $T[1, i]$ 是否循环同构。
- 判断相等只需要比较 Hash 值是否相等。
- 问题的关键在于判断 S 和 T 的每一个前缀是否循环同构。

60 分做法

- 引理：如果 x 和 y 循环同构，那么 $x = uv, y = vu$。

60 分做法

- 引理：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 $C(x, y) = x_1 y_n x_2 y_{n-1} \cdots x_{n-1} y_2 x_n y_1$。

60 分做法

- 引理：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。

60 分做法

- 引理：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。

60 分做法

- 引理：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。

60 分做法

- 引理：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。

60 分做法

- 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。

60 分做法

- 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。

60 分做法

- 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。
- 考虑 $C(x, y)$ 的定义，我们可以得到一个更强的结论：

60 分做法

- 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。
- 考虑 $C(x, y)$ 的定义，我们可以得到一个更强的结论：
- 若 S 和 T 循环同构，那么 $S = uv, T = vu$，要么 u, v 至少有一个是 S 和 T 最长匹配的前后缀。

60 分做法

- 回到一开始的问题，依次枚举每个前缀 i，判断 $S[1,i]$ 和 $T[1,i]$ 是否循环同构。

60 分做法

- 回到一开始的问题，依次枚举每个前缀 i，判断 $S[1,i]$ 和 $T[1,i]$ 是否循环同构。
- 由于只需要求最长匹配的前后缀，因此在枚举的过程中进行 KMP 即可，KMP 的指针位置就是最长匹配的前后缀。

60 分做法

- 回到一开始的问题，依次枚举每个前缀 i，判断 $S[1,i]$ 和 $T[1,i]$ 是否循环同构。
- 由于只需要求最长匹配的前后缀，因此在枚举的过程中进行 KMP 即可，KMP 的指针位置就是最长匹配的前后缀。
- 对于剩下的部分，只需要 Hash 判断两个子串是否相等。

60 分做法

- $2.S = ABC, T = ACB$。

60 分做法

- 2.$S = ABC$, $T = ACB$。
- 将 S 和 T 都翻转，那么就转化为了情况 1。

60 分做法

- $3.S = ABC, T = CBA$。

60 分做法

- 3.$S = ABC$, $T = CBA$。
- 此时 $C(S, T)$ 可以被拆分为 3 个长度为偶数的回文串。

60 分做法

- 3.$S = ABC$, $T = CBA$。
- 此时 $C(S, T)$ 可以被拆分为 3 个长度为偶数的回文串。
- 枚举 $C(S, T)$ 的每一个前缀，判断是否可以拆分为两个偶回文串，那么只需要判断后面部分是否是回文串即可。

60 分做法

- 3.$S = ABC$, $T = CBA$。
- 此时 $C(S, T)$ 可以被拆分为 3 个长度为偶数的回文串。
- 枚举 $C(S, T)$ 的每一个前缀，判断是否可以拆分为两个偶回文串，那么只需要判断后面部分是否是回文串即可。
- 利用 Hash 可以进行回文的判定以及最长回文前缀的求解。

60 分做法

- 3.$S = ABC$, $T = CBA$。
- 此时 $C(S, T)$ 可以被拆分为 3 个长度为偶数的回文串。
- 枚举 $C(S, T)$ 的每一个前缀，判断是否可以拆分为两个偶回文串，那么只需要判断后面部分是否是回文串即可。
- 利用 Hash 可以进行回文的判定以及最长回文前缀的求解。
- 一个前缀的最长偶回文后缀可以用 Palindromic Tree 不假思索地完成。

60 分做法

- 3.$S = ABC$, $T = CBA$。
- 此时 $C(S, T)$ 可以被拆分为 3 个长度为偶数的回文串。
- 枚举 $C(S, T)$ 的每一个前缀，判断是否可以拆分为两个偶回文串，那么只需要判断后面部分是否是回文串即可。
- 利用 Hash 可以进行回文的判定以及最长回文前缀的求解。
- 一个前缀的最长偶回文后缀可以用 Palindromic Tree 不假思索地完成。
- 时间复杂度 $O(n \log n)$。常数优秀可以得到 100 分。

100 分做法

- 60 分做法的瓶颈在于求每个前缀的最长偶回文后缀。

100 分做法

- 60 分做法的瓶颈在于求每个前缀的最长偶回文后缀。
- 利用 Manacher 算法可以得到以每个位置为中心的最长回文半径。

100 分做法

- 60 分做法的瓶颈在于求每个前缀的最长偶回文后缀。
- 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。

100 分做法

- 60 分做法的瓶颈在于求每个前缀的最长偶回文后缀。
- 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。

100 分做法

- 60 分做法的瓶颈在于求每个前缀的最长偶回文后缀。
- 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。

100 分做法

- 60 分做法的瓶颈在于求每个前缀的最长偶回文后缀。
- 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 $O(n)$。

确定性算法

- 能否不用 Hash 来判断两个子串是否相等？

确定性算法

- 能否不用 Hash 来判断两个子串是否相等？
- 答案是肯定的。

确定性算法

- 能否不用 Hash 来判断两个子串是否相等？
- 答案是肯定的。
- 注意到每次涉及比较的串中至少有一个串是某个串的前缀或者后缀。

确定性算法

- 能否不用 Hash 来判断两个子串是否相等？
- 答案是肯定的。
- 注意到每次涉及比较的串中至少有一个串是某个串的前缀或者后缀。
- 因此只要预处理出每个串与每个串每个前后缀的最长公共前缀即可。

确定性算法

- 能否不用 Hash 来判断两个子串是否相等？
- 答案是肯定的。
- 注意到每次涉及比较的串中至少有一个串是某个串的前缀或者后缀。
- 因此只要预处理出每个串与每个串每个前后缀的最长公共前缀即可。
- 利用扩展 KMP 算法就可以解决这个问题。

确定性算法

- 能否不用 Hash 来判断两个子串是否相等？
- 答案是肯定的。
- 注意到每次涉及比较的串中至少有一个串是某个串的前缀或者后缀。
- 因此只要预处理出每个串与每个串每个前后缀的最长公共前缀即可。
- 利用扩展 KMP 算法就可以解决这个问题。
- 时间复杂度 $O(n)$。

Thank you!