J - lastendconductor

题目描述

定义一个字符串 T 是**周期的**，当且仅当存在字符串 T' 使得 T 可由**至少** 2 **个** T' 依次连接得到。

例如：

- $T_1 =$ `gamegame` 是周期的，因为 $T_1 =$ `gamegame` 可由 2 个 $T_1' =$ `game` 连接得到；
- $T_2 =$ `qwqqwqqwq` 也是周期的，因为 $T_2 =$ `qwqqwqqwq` 可由 3 个 $T_2' =$ `qwq` 连接得到；
- $T_3 =$ `lastendconductor` 不是周期的，因为不存在满足上述条件的 T_3'。

对于字符串 $T = T_1 T_2 \cdots T_{|T|}$，我们称 T' 是 T 的**子序列**，当且仅当存在下标数列 $1 \le p_1 < p_2 < \cdots < p_{|T'|} \le |T|$ 使得 $T' = T_{p_1} T_{p_2} \cdots T_{p_{|T'|}}$。

给定仅包含小写英文字母的字符串 S，请计算 S 的所有**子序列**中周期字符串的最长长度。如果 S 的子序列中不存在周期字符串，则认为答案为 0。

最长公共子序列

P,Q,R为长度分别为o,p,q的字符串，

P,Q的最长公共子序列长度为：

$$LCS(P,Q) = \begin{cases} LCS(P_{1\dots o-1}，Q_{1\dots p-1}) & P_o = Q_p \\ max\{LCS(P_{1\dots o-1},Q),LCS(P,Q_{1\dots p-1})\} & P_o \ne Q_p \end{cases}$$

若$P_o = Q_p$则P_o,Q_p一定在LCS中，否则P_o在LCS中或者P_o与$Q_k(1 \le k < p)$同时在LCS中都不会使答案变优。

若$P_o \ne Q_p$，必有一者不在LCS中

可以扩展为三个字符串的情况：

P,Q,R的最长公共子序列长度为：

$$LCS(P,Q,R) = \begin{cases} LCS(P_{1\dots o-1}，Q_{1\dots p-1},R_{1\dots q-1}) & P_o = Q_p = R_q \\ max\{LCS(P_{1\dots o-1}，Q,R),LCS(P，Q_{1\dots p-1},R),LCS(P，Q,R_{1\dots q-1})\} & else \end{cases}$$

观察1：

对于偶数周期数的周期子序列，$U = U_1U_2\dots U_{2m}(U_i = U_j)$

- 可知$U_1 \dots U_m = U_{n+1} \dots U_{2m}$，同时有$U_1 \dots U_m$是$S_{1\dots k}$的子序列，$U_{m+1} \dots U_{2m}$是$S_{k+1\dots n}$的子序列。$U_1 \dots U_m$，即是$S_{1\dots k}$和$S_{k+1\dots n}$的最长公共子序列。
- 反之，对于$1 \le k \le |S|$，$S_{1\dots k}$和$S_{k+1\dots n}$的公共子序列拼接后一定是周期数为偶数的周期子串。

周期数为偶数的最长周期子序列 $\iff \max_{1\le k<n}\{LCS(S_{1\dots k},S_{k+1\dots n})\}$

这种情况下可以通过枚举k在$O(n^3)$的复杂度下算出。

观察2：

我们需要考虑周期数为奇数的周期子序列$U = U_1 \dots U_m$

- $3|m$时，可以使用与观察1类似的思想，将S分为三段，找出三个子串的LCS

  周期数为$3x$的最长周期序列 $\iff \max_{1\le l<r<n}\{LCS(S_{1\dots l},S_{l+1\dots r},S_{r+1\dots n})\}$

  这种情况下可以通过枚举l,r在$O(n^5)$的复杂度上算出

  实际上$n = 80$时计算量约为$\Sigma_{1\le l<r<80} l(r-l)(n-r) = 27285336$

```
>>> for i in range(1,80):
        for j in range(i+1,80):
                ans=ans+i*(j-i)*(80-j)


>>> print(ans)
27285336
>>>
```

- 否则，有$m\geq 5$，使用反证法可以证明，$\exists 1\leq i\leq m$，使得子序列U_i在原串的长度$\leq\lfloor n/5\rfloor\leq 16$，如果对每一个长度$\leq\lfloor n/5\rfloor$的子串进行子序列枚举，可以贪心地在原串中查找子序列，时间复杂为$O(n)$，总的时间复杂度为$O(2^{\lfloor n/5\rfloor -1}n^2)$（指数-1的原因是搜索时规定长度$\leq\lfloor n/5\rfloor$的子串中第一个字符必选），实际上还可以再进行优化（虽然可以混过去）。

观察3：

对于$m\geq 5$周期数的周期子序列。

把S均分成5段后，如果每个U_i都跨越了分界点，$1\leq i<j\leq n$时，U_i和U_j不会同时跨越一个分界点，因而每个分界点只能被一个U_i跨越，因此只会出现4个循环节，与前提矛盾，故一定有一个循环节在均分后的某一段内。

于是，只需要枚举5段子串的子序列，再贪心地进行在原串中查找周期。这种情况下时间复杂度为$O(2^{\lceil n/5\rceil}n)$

综合以上的三种情况，

总的时间复杂度为$O(n^5+2^{\lceil n/5\rceil}n)$.

代码：

```cpp
#include <bits/stdc++.h>

using namespace std;

int t, n, sslen;
char s[90];
int dp2[90][90], dp3[90][90][90];
char subs[90];

int lcs2(char* s1, char* s2, int n, int m) {
    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= m; ++j) {
            if (s1[i - 1] == s2[j - 1])
                dp2[i][j] = dp2[i - 1][j - 1] + 1;
            else
                dp2[i][j] = max(dp2[i - 1][j], dp2[i][j - 1]);
        }
    }
    return dp2[n][m];
}

int lcs3(char* s1, char* s2, char* s3, int n, int m, int o) {
    if (n * m * o == 0) return 0;
    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= m; ++j) {
            for (int k = 1; k <= o; ++k) {
                if (s1[i - 1] == s2[j - 1] && s2[j - 1] == s3[k - 1]) {
                    dp3[i][j][k] = dp3[i - 1][j - 1][k - 1] + 1;
```

```
                } else {
                    dp3[i][j][k] = max(dp3[i - 1][j][k], max(dp3[i][j - 1][k], dp3[i][j][k
- 1]));
                }
            }
        }
    }
    return dp3[n][m][o];
}

int occurence() {
    int j = 0;
    int ans = 0;
    for (int i = 0; s[i]; ++i) {
        if (s[i] == subs[j]) {
            ++j;
            if (j == sslen) {
                j = 0;
                ++ans;
            }
        }
    }
    return ans;
}

int main() {
    cin.tie(NULL);
    ios::sync_with_stdio(false);

    cin >> t;
    while (t--) {
        cin >> s;
        n = 0;
        while (s[n]) ++n;
        int ans = 0;
        for (int i = 1; i <= n; ++i) {
            ans = max(ans, 2 * lcs2(s, s + i, i, n - i));
        }

        for (int i = 1; i < n; ++i) {
            for (int j = 1; i + j < 80; ++j) {
                int k = 80 - i - j;
                ans = max(ans, 3 * lcs3(s, s + i, s + i + j, i, j, k));
            }
        }

        int elen = (n + 4) / 5;

        for (int mask = 0; mask < (1 << elen); ++mask) {
            for (int st = 0; st < n; st += elen) {
                sslen = 0;
                for (int i = 0; i < elen && st + i + 1 < n; ++i) {
                    if ((mask >> i) & 1) {
                        subs[sslen] = s[st + i];
                        ++sslen;
                    }
                }
                int oc = occurence();
                if (oc > 1)
                    ans = max(ans, oc * sslen);
            }
```

```
        }

        cout << ans << endl;
    }
    return 0;
}
```