P4421 [COCI 2017/2018 #1] Lozinke

题意

给定一个字符串 U，它是由字符串 S 经过以下操作得到的：先将 S 复制两份得到 T（即 $T = S + S$），然后在 T 的任意位置（包括首尾）插入一个字符得到 U。要求根据 U 还原出原始的字符串 S。如果存在多个可能的 S，则输出 "NOT UNIQUE"；如果无法还原，则输出 "NOT POSSIBLE"。

数据范围：字符串 U 的长度 n 满足 $1 \leq n \leq 2 \times 10^6$，且 n 为奇数（因为插入一个字符后长度为奇数）。

分析

由于 U 是由 S 复制两次后插入一个字符得到的，因此 U 的长度 n 必为奇数。如果 n 为偶数，直接输出 "NOT POSSIBLE"。对于奇数 n，设 $mid = \lfloor n/2 \rfloor$，则 S 的长度应为 mid。算法核心是枚举插入字符的位置，并利用字符串哈希快速比较删除该字符后剩余部分是否能分成两个相等的子串（即 S）。

具体步骤：

1. 预处理字符串 U 的哈希值，使用基数 $base = 22783$ 和自然溢出（unsigned long long）作为模数。
2. 计算前缀哈希数组 $h[i]$ 和基数幂数组 $pre[i]$，使得可以在 $O(1)$ 时间内计算任意子串的哈希值。

3. 枚举插入字符的位置 k：
    - 如果 $k \leq mid$，则删除 $U[k]$ 后，前 mid 个字符（跳过 k）应与后 mid 个字符匹配。
    - 如果 $k > mid$，则删除 $U[k]$ 后，前 mid 个字符应与后 mid 个字符（跳过 k）匹配。
4. 使用哈希值比较匹配情况，记录所有可行的 S。如果找到多个不同的 S，输出 "NOT UNIQUE"；如果只有一个，输出该 S；如果没有，输出 "NOT POSSIBLE"。

时间复杂度：$O(n)$，因为预处理哈希值需要 $O(n)$，枚举每个位置并检查需要 $O(1)$ 时间。

参考代码

```
#define ll unsigned long long
const int N = 20005;
unordered_map<ll, int> _hash, _hash2; // 整串的hash，串是否已出现
ll vis[N];                            // 记录子串 hash，用于避免重复计算贡献
string s[N];
int n;
ll get(string t, int l, int r) {
    ll _hashcode = 1;
    for (int i = l; i <= r; i++)
        _hashcode = _hashcode * 26 + (t[i] - 'a') + 1;
    return _hashcode;
}
void solve() {
    for (int i = 1; i <= n; i++)
        _hash[get(s[i], 0, s[i].size() - 1)]++;
}
```

```
ll check(string t) {
    ll ans = -1; /* 0~0 是空串肯定存在，则预-1 */
    int len = t.size(), cnt = 0;
    /* 暴力枚举子串 */
    for (int i = 0; i < len; i++)
        for (int j = i; j < len; j++) {
            /* 子串存在 && 子串没有计算过贡献 */
            ll _subhashcode = get(t, i, j);
            if (_hash[_subhashcode] && !_hash2[_subhashcode]) {
                ans += _hash[_subhashcode];
                _hash2[_subhashcode] = 1, vis[++cnt] = _subhashcode;
            }
        }
    /* 还原占位 */
    for (int i = 1; i <= cnt; i++)
        _hash2[vis[i]] = 0;
    return ans;
}
int main() {
    cin >> n;
    for (int i = 1; i <= n; i++)
        cin >> s[i];
    solve();
    ll ans = 0;
    for (int i = 1; i <= n; i++)
        ans += check(s[i]);
    cout << ans << endl;
    return 0;
}
```

P6739 【BalticOI 2014 Day1】Three Friends

题意

给定一个字符串 U，它是由字符串 S 经过以下操作得到的：先将 S 复制两份得到 T（即 $T = S + S$），然后在 T 的任意位置（包括首尾）插入一个字符得到 U。要求根据 U 还原出原始的字符串 S。如果存在多个可能的 S，则输出 "NOT UNIQUE"；如果无法还原，则输出 "NOT POSSIBLE"。

数据范围：字符串 U 的长度 n 满足 $1 \leq n \leq 2 \times 10^6$，且 n 为奇数（因为插入一个字符后长度为奇数）。

分析

核心思路是使用字符串哈希技术。字符串哈希将字符串视为一个基进制数，通过计算哈希值来快速比较字符串是否相等。具体步骤：

1. **预处理**：计算前缀哈希数组 $h[i]$ 和基的幂数组 $pre[i]$，基取 22783，使用 `unsigned long long` 自然溢出作为模数。
2. **特判**：如果 n 为偶数，直接输出 "NOT POSSIBLE"，因为插入字符后长度必为奇数。

3. **枚举插入位置**：设中点 $mid = \lfloor n/2 \rfloor + 1$。枚举插入位置 k：
   - 如果 $k \leq mid$，则删除 $U[k]$ 后，左半部分（跳过 k）的哈希值应与右半部分哈希值相等。
   - 如果 $k > mid$，则删除 $U[k]$ 后，右半部分（跳过 k）的哈希值应与左半部分哈希值相等。
4. **结果判断**：根据匹配情况统计可能解的数量。如果只有一个解，输出 S；如果多个解但 S 相同，输出 S；如果多个解且 S 不同，输出 "NOT UNIQUE"；无解输出 "NOT POSSIBLE"。

时间复杂度：预处理 $O(n)$，枚举 $O(n)$，总时间复杂度 $O(n)$。

参考代码

```
#define ull unsigned long long
const int N = 2e6 + 10;
ull base = 13331; // 哈希基数
int n, mid, len1, len2, ans;
string s, a, b, c, d;
ull hsh[N], pre[N]; // hsh: 前缀哈希数组, pre: 基数幂次数组

// 获取区间[l, r]的哈希值
ull getHash(int l, int r) { return hsh[r] - hsh[l - 1] * pre[r - l + 1]; }

// 获取区间[l, r]中删除位置k后的哈希值
ull getSubHash(int l, int r, int k) {
    // 将[l, k-1]和[k+1, r]两段拼接起来计算哈希
    return getHash(l, k - 1) * pre[r - k] + getHash(k + 1, r);
}
```

```cpp
int main() {
    ios::sync_with_stdio(false), cin.tie(nullptr);
    cin >> n >> s;

    // 检查字符串长度是否为奇数，偶数长度不可能通过删除一个字符形成回文
    if (!(n % 2))
        puts("NOT POSSIBLE"), exit(0);

    s = " " + s;                        // 让字符串下标从1开始
    pre[0] = 1, mid = (n + 1) >> 1; // mid是中间位置

    // 预处理哈希数组和基数幂次数组
    for (int i = 1; i <= n; i++) {
        hsh[i] = hsh[i - 1] * base + (s[i] - 'A' + 1);
        pre[i] = pre[i - 1] * base;
    }

    // 情况1：删除字符在左半部分
    len1 = getHash(mid + 1, n);      // 右半部分的哈希值
    a = s.substr(mid + 1, n - mid); // 右半部分的字符串

    // 遍历左半部分的每个位置，尝试删除该字符
    for (int i = 1; i <= mid; i++) {
        // 计算删除位置i后左半部分的哈希值
        len2 = getSubHash(1, mid, i);
        if (len1 == len2) { // 如果左右两半哈希值相等
            ans++;
            c = a; // 记录结果字符串
            break; // 找到一个解就退出
        }
    }
```

```
    // 情况2：删除字符在右半部分
    len2 = getHash(1, mid - 1); // 左半部分(不含中间字符)的哈希值
    b = s.substr(1, mid - 1);   // 左半部分的字符串

    // 遍历右半部分的每个位置，尝试删除该字符
    for (int i = mid; i <= n; i++) {
        // 计算删除位置i后右半部分的哈希值
        len1 = getSubHash(mid, n, i);
        if (len1 == len2) { // 如果左右两半哈希值相等
            ans++;
            d = b; // 记录结果字符串
            break; // 找到一个解就退出
        }
    }

    // 输出结果
    if (ans == 0)
        puts("NOT POSSIBLE"); // 无解
    else if (ans == 1 || c == d)
        cout << (c.size() ? c : d) << endl; // 唯一解或两个解相同
    else
        puts("NOT UNIQUE"); // 多个不同解

    return 0;
}
```