重复的DNA序列 题解

> 计算学部十大打卡活动——“龙舞编程新春会”编程打卡（2024-1-25）
>
> 力扣187.重复的DNA序列

一、题目

DNA序列 由一系列核苷酸组成，缩写为 'A', 'C', 'G' 和 'T'.。

- 例如，"ACGAATTCCG" 是一个 **DNA序列** 。

在研究 **DNA** 时，识别 DNA 中的重复序列非常有用。

给定一个表示 **DNA序列** 的字符串 s ， 返回所有在DNA分子中出现不止一次的 **长度为** 10 的序列(子字符串)。你可以按 **任意顺序** 返回答案。

示例 1：

> 输入：s = "AAAAACCCCCAAAAACCCCCCAAAAAGGGTTT"
>
> 输出：["AAAAACCCCC","CCCCCAAAAA"]

示例 2：

> 输入：s = "AAAAAAAAAAAAA"
>
> 输出：["AAAAAAAAAA"]

提示：

- 0 <= s.length <= 10^5
- s[i]=='A'、'C'、'G' or 'T'

二、题解

方法一 哈希表

思路

可以使用哈希表统计 s 中所有长度为 10 的子串的出现次数，返回所有出现不止一次的子串。

代码实现时，可以一边遍历子串一边记录答案，为了不重复记录答案，只统计当前出现次数为 2 的子串。

代码

```
class Solution {
    const int L = 10;

public:
    vector<string> findRepeatedDnaSequences(string s) {
        vector<string> ans;
        unordered_map<string, int> cnt;
        int n = s.length(), i;
        for (i = 0; i <= n - L; i++) {
            string sub = s.substr(i, L); //提取子串
            if (++cnt[sub] == 2) {
                ans.push_back(sub);
            }
        }
        return ans;
    }
};
```

复杂度分析

- 时间复杂度：$O(NL)$，其中 N 是字符串 s 的长度，$substr()$ 需要的时间为 $O(L)$ 。
- 空间复杂度：$O(NL)$。

方法二 哈希表+滑动窗口+位运算

思路

由于 s 中只含有 4 种字符，可以将每个字符用 2 个比特表示，即：

- `A` 表示为二进制 `00`；
- `C` 表示为二进制 `01`；
- `G` 表示为二进制 `10`；
- `T` 表示为二进制 `11`；

所以，一个长为 10 的字符串就可以用 20 个比特表示，而一个 `int` 整数有 32 个比特，足够容纳该字符串，因此可以将 s 的每个长为 10 的子串用一个 `int` 整数表示（只用低 20 位）。

同时注意到每个整数都对应唯一一个字符串，可以将方法一中的哈希表改为存储每个整数的个数。

若我们对每个子串都单独计算其整数表示，则时间复杂度仍然和方法一一样。为了优化时间复杂度，可以用一个大小固定为 10 的滑动窗口来计算子串的整数表示。设当前滑动窗口对应的整数表示为 x，当我们要计算下一个子串时，就将滑动窗口向右移动一位，此时会有一个新的字符进入窗口，以及窗口最左边的字符离开窗口，这些操作对应的位运算，按计算顺序表示如下：

- 由于每个字符用 2 个比特表示，滑动窗口向右移动一位，则 x 要左移 2 位：`x = x << 2`；
- 一个新的字符 ch 进入窗口：`x = x | bin[ch]`，这里 $bin[ch]$ 为字符 ch 对应二进制；
- 由于我们只考虑 x 的低 20 位比特，所以将 x 左移 2 位后，还需对高位置零，即与上 `(1 << 20) - 1`：`x = x & ((1 << 20) - 1)`。

将这三步合并，就可以用 $O(1)$ 的时间计算出下一个子串的整数表示，即 `x = ((x << 2) | bin[ch]) & ((1 << 20) - 1)`。

代码

```
class Solution {
    const int L = 10;
    unordered_map<char, int> bin = {{'A', 0}, {'C', 1}, {'G', 2}, {'T', 3}};

public:
    vector<string> findRepeatedDnaSequences(string s) {
        vector<string> ans;
        unordered_map<int, int> cnt;
        int n = s.length(), i, x = 0;
        //若s长度小于等于10，则直接返回空
        if (n <= L) {
            return ans;
        }
        //初始化
        for (i = 0; i < L - 1; i++) {
            x = (x << 2) | bin[s[i]];
        }
        for (i = 0; i <= n - L; i++) {
            x = ((x << 2) | bin[s[i + L - 1]]) & ((1 << (L * 2)) - 1);
            if (++cnt[x] == 2) {
                ans.push_back(s.substr(i, L));
            }
        }
        return ans;
    }
};
```

复杂度分析

- 时间复杂度：$O(N)$，其中 N 是字符串 s 的长度。
- 空间复杂度：$O(N)$。