Repeated DNA Sequences

All DNA is composed of a series of nucleotides abbreviated as A, C, G, and T, for example: "ACGAATTCCG". When studying DNA, it is sometimes useful to identify repeated sequences within the DNA.

Write a function to find all the 10-letter-long sequences (substrings) that occur more than once in a DNA molecule.

For example,

```
Given s = "AAAAACCCCCAAAAACCCCCCAAAAAGGGTTT",

Return:
["AAAAACCCCC", "CCCCCAAAAA"].
```

Solution 1

The main idea is to store the substring as int in map to bypass the memory limits.

There are only four possible character A, C, G, and T, but I want to use 3 bits per letter instead of 2.

Why? It's easier to code.

A is 0x41, C is 0x43, G is 0x47, T is 0x54. Still don't see it? Let me write it in octal.

A is 0101, C is 0103, G is 0107, T is 0124. The last digit in octal are different for all four letters. That's all we need!

We can simply use `s[i] & 7` to get the last digit which are just the last 3 bits, it's much easier than lookup table or switch or a bunch of if and else, right?

We don't really need to generate the substring from the int. While counting the number of occurrences, we can push the substring into result as soon as the count becomes 2, so there won't be any duplicates in the result.

```
vector<string> findRepeatedDnaSequences(string s) {
    unordered_map<int, int> m;
    vector<string> r;
    int t = 0, i = 0, ss = s.size();
    while (i < 9)
        t = t << 3 | s[i++] & 7;
    while (i < ss)
        if (m[t = t << 3 & 0x3FFFFFFF | s[i++] & 7]++ == 1)
            r.push_back(s.substr(i - 10, 10));
    return r;
}
```

BTW, the OJ doesn't seems to have test cases which the given string length is smaller than 9, so I didn't check it to make the code simpler.

Any suggestions?

Update:

I realised that I can use `s[i] >> 1 & 3` to get 2 bits, but then I won't be able to remove the first loop as 1337c0d3r suggested.

written by sen original link here

Solution 2

```
public List<String> findRepeatedDnaSequences(String s) {
    Set<Integer> words = new HashSet<>();
    Set<Integer> doubleWords = new HashSet<>();
    List<String> rv = new ArrayList<>();
    char[] map = new char[26];
    //map['A' - 'A'] = 0;
    map['C' - 'A'] = 1;
    map['G' - 'A'] = 2;
    map['T' - 'A'] = 3;

    for(int i = 0; i < s.length() - 9; i++) {
        int v = 0;
        for(int j = i; j < i + 10; j++) {
            v <<= 2;
            v |= map[s.charAt(j) - 'A'];
        }
        if(!words.add(v) && doubleWords.add(v)) {
            rv.add(s.substring(i, i + 10));
        }
    }
    return rv;
}
```

written by crazyirontoiletpaper original link here

Solution 3

Hi guys!

The idea is to use rolling hash technique or in case of string search also known as Rabin-Karp algorithm. As our alphabet A consists of only 4 letters we can be not afraid of collisions. The hash for a current window slice could be found in a constant time by subtracting the former first character times size of the A in the power of 9 and updating remaining hash by the standard rule: hash = hash*A.size() + curr_char.

Check out the Java code below.

Hope it helps!

```
public class Solution {
    private static final Map<Character, Integer> A = new HashMap<>();
    static { A.put('A',0); A.put('C',1); A.put('G',2); A.put('T',3); }
    private final int A_SIZE_POW_9 = (int) Math.pow(A.size(), 9);

    public List<String> findRepeatedDnaSequences(String s) {
        Set<String> res = new HashSet<>();
        Set<Integer> hashes = new HashSet<>();
        for (int i = 0, rhash = 0; i < s.length(); i++) {
            if (i > 9) rhash -= A_SIZE_POW_9 * A.get(s.charAt(i-10));
            rhash = A.size() * rhash + A.get(s.charAt(i));
            if (i > 8 && !hashes.add(rhash)) res.add(s.substring(i-9,i+1));
        }
        return new ArrayList<>(res);
    }
}
```

written by shpolsky original link here