Leetcode\_187\_RepeatedDNASequences\_重复的DAN序列\_Medium

# Leetcode\_187\_RepeatedDNASequences\_重复的DAN序列\_Medium

## 题目介绍

\* 难度：Medium

\* 题目介绍：

\* <https://leetcode.com/problems/repeated-dna-sequences/description/>

\* 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.

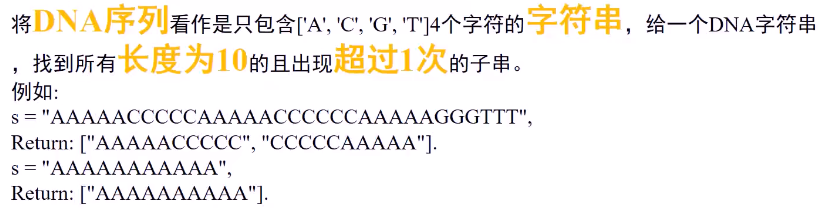
\* Write a function to find all the 10-letter-long sequences (substrings)

\* that occur more than once in a DNA molecule.

\* Example:

\* Input: s = "AAAAACCCCCAAAAACCCCCCAAAAAGGGTTT"

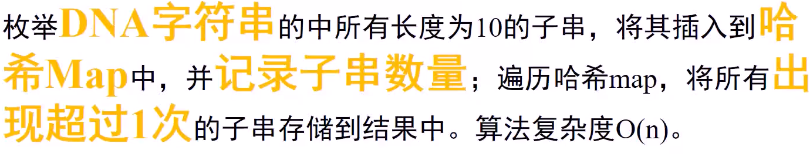
\* Output: ["AAAAACCCCC", "CCCCCAAAAA"]



## 思路分析

### 思路1：枚举方法。

思路1：与DNA序列无关；十分简单的做法。直接利用10个字符的字符串作为HashMap的key，出现次数为value；最终遍历将出现次数大于1次的添加到list中即可。

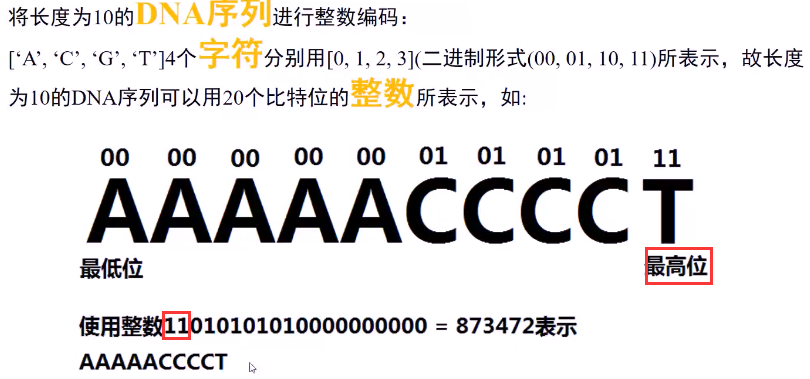


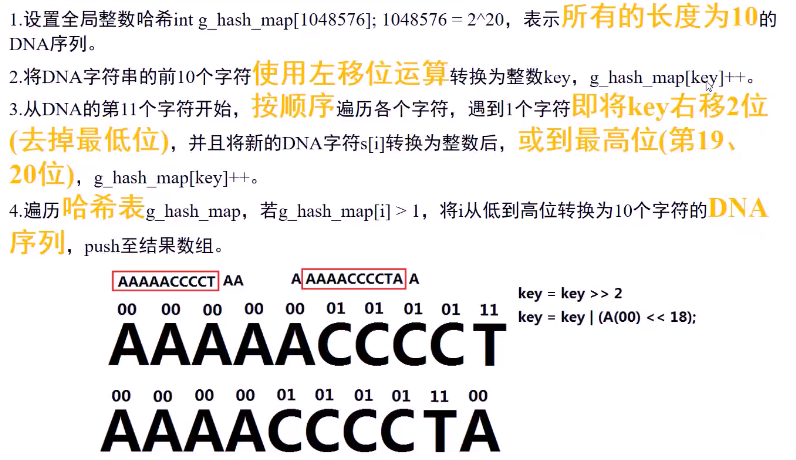
### 思路2：编码，位运算。

\* 思路2:尽量两用DNA特点。由于DNA字符只有ACGT，因此使用编码的方式，

\* 分别用00,01,10,11表示四个字符，构造一个占用20位的数字，构建哈希表；

\* 利用位运算实现移动；最后也是遍历哈希表，输出结果。





## Java代码

### 思路1代码：

//思路1的实现

public List<String> findRepeatedDnaSequences1(String s) {

List<String> result = new ArrayList<String>();

if(s == null||s.length() <= 9) return result;//边界条件

HashMap<String,Integer> hashMap = new HashMap<String,Integer>();//HashMap

for(int i = 0;i <= s.length() -10;i++) {//注意是<=

String key = s.substring(i, i + 10);

if (hashMap.containsKey(key)) hashMap.put(key, hashMap.get(key) + 1);

else hashMap.put(key, 1);

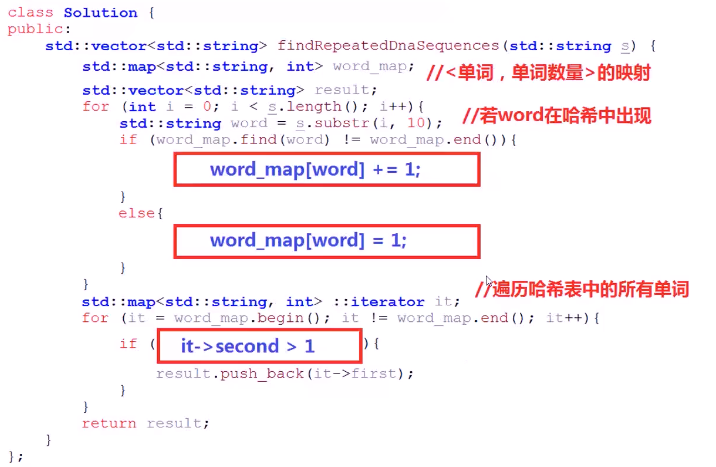
}

for(String key : hashMap.keySet())

if (hashMap.get(key) > 1) result.add(key);

return result;

}



### 思路2：位运算，效率高

/\*\*

\* 方法2：编码,解码,位运算

\*/

public static int MAX\_VALUE = 0X100000;//20位全1加1

public static int MAX\_VALUE\_1 = 0X100000-1;//20位全1

public List<String> findRepeatedDnaSequences(String s) {

List<String> result = new ArrayList<String>();

if (s == null || s.length() <= 9) return result;

char[] chs = s.toCharArray();

int[] count = new int[MAX\_VALUE];

int key = 0;

for(int i = 0;i < 10;i++){//初始第一个key

key = (key << 2)|coding(chs[i]);

}

count[key]++;

for(int i = 10;i < chs.length;i++){//遍历一遍

key = (key <<2)|coding(chs[i]);//向右移

key &= MAX\_VALUE\_1;//将高位置0

count[key]++;

}

//找到出现两次及之上的DNA序列

for(int i = 0;i < MAX\_VALUE;i++)

if(count[i] > 1) result.add(decoding(i));

return result;

}

/\*\*

\* 编码：将字符转换成00、01、10、11

\*/

private int coding(char c){

switch(c){

case 'A':return 0;

case 'C':return 1;

case 'G':return 2;

default:return 3;

}

}

/\*\*

\* 解码：将数字key转成字符串

\*/

private String decoding(int key){

char[] deCoding = {'A','C','G','T'};

char[] tenChars = new char[10];

for(int j = 9;j>=0;j--) {

tenChars[j] = deCoding[key&3];

key = key>>2;

}

return new String(tenChars);

}

