重复的子串

2022年9月2日 11:41

459. 重复的子字符串

难度 简单 769

给定一个非空的字符串 `s` ，检查是否可以通过由它的一个子串重复多次构成。

示例 1:

```
输入: s = "abab"
输出: true
解释: 可由子串 "ab" 重复两次构成。
```

示例 2:

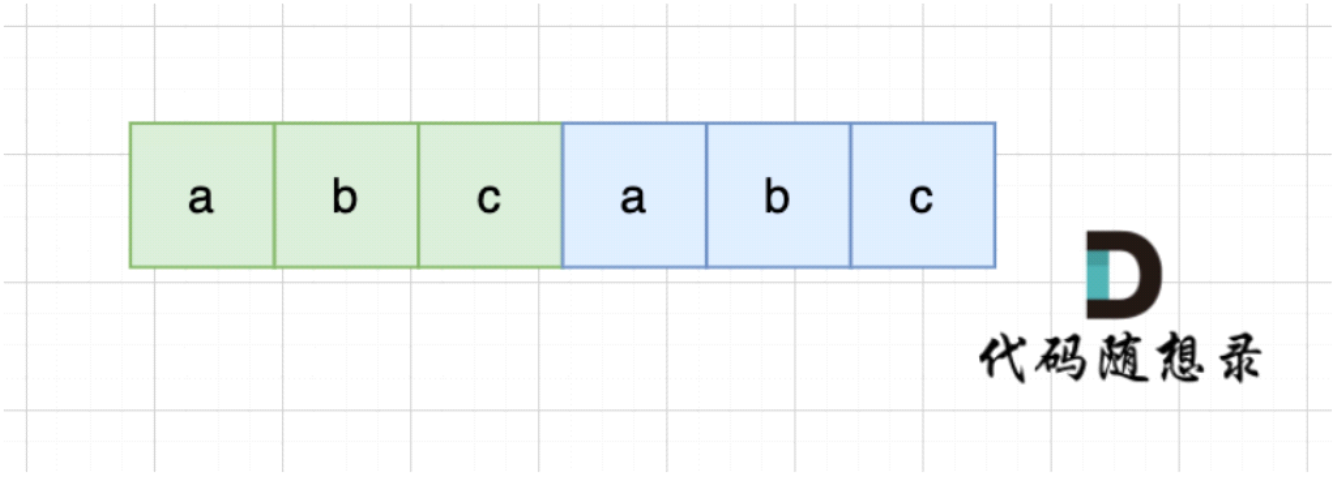
```
输入: s = "aba"
输出: false
```

若此题采用暴力解法，则通过一个for循环获取子串的终止位置，然后又嵌套一个for循环，判断子串是否能重复构成字符串。
时间复杂度为o(n^2)
注意只需要获取子串的终止位置而无需起始位置是因为该题判断一个字符串s能否通过子串重复多次构成，若子串不从位置0处的元素开始，也就没有意义了。

移动匹配

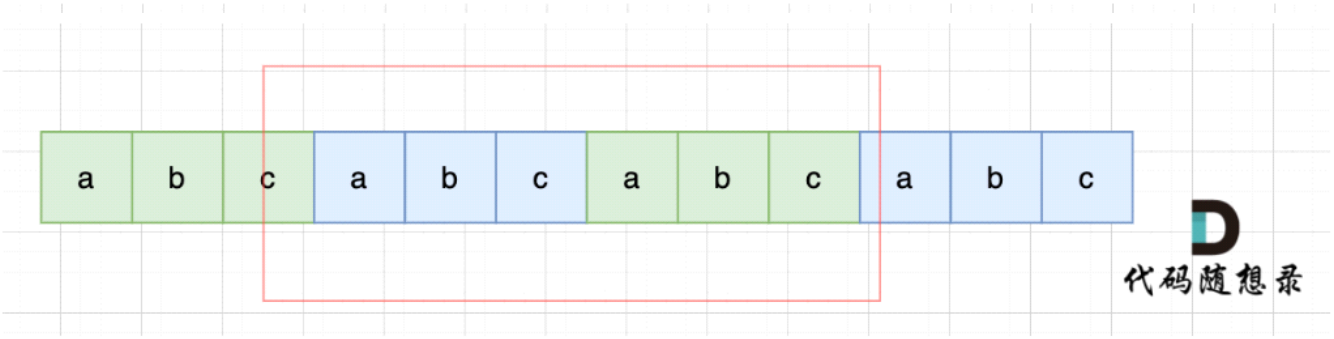
核心思想：若一个字符串s由重复子串构成，那么当两个s拼接在一起时（s+s），一定能从中间部分重新找到s（中间即不包括头，也不包括尾部）。

当一个字符串s：abcabc，内部又重复的子串组成，那么这个字符串的结构一定是这样的：



也就是又前后又相同的子串组成。

那么既然前面有相同的子串，后面有相同的子串，用 s + s，这样组成的字符串中，后面的子串做前串，前后的子串做后串，就一定还能组成一个s，如图：



所以判断字符串s是否有重复子串组成，只要两个s拼接在一起，里面还出现一个s的话，就说明是又重复子串组成。

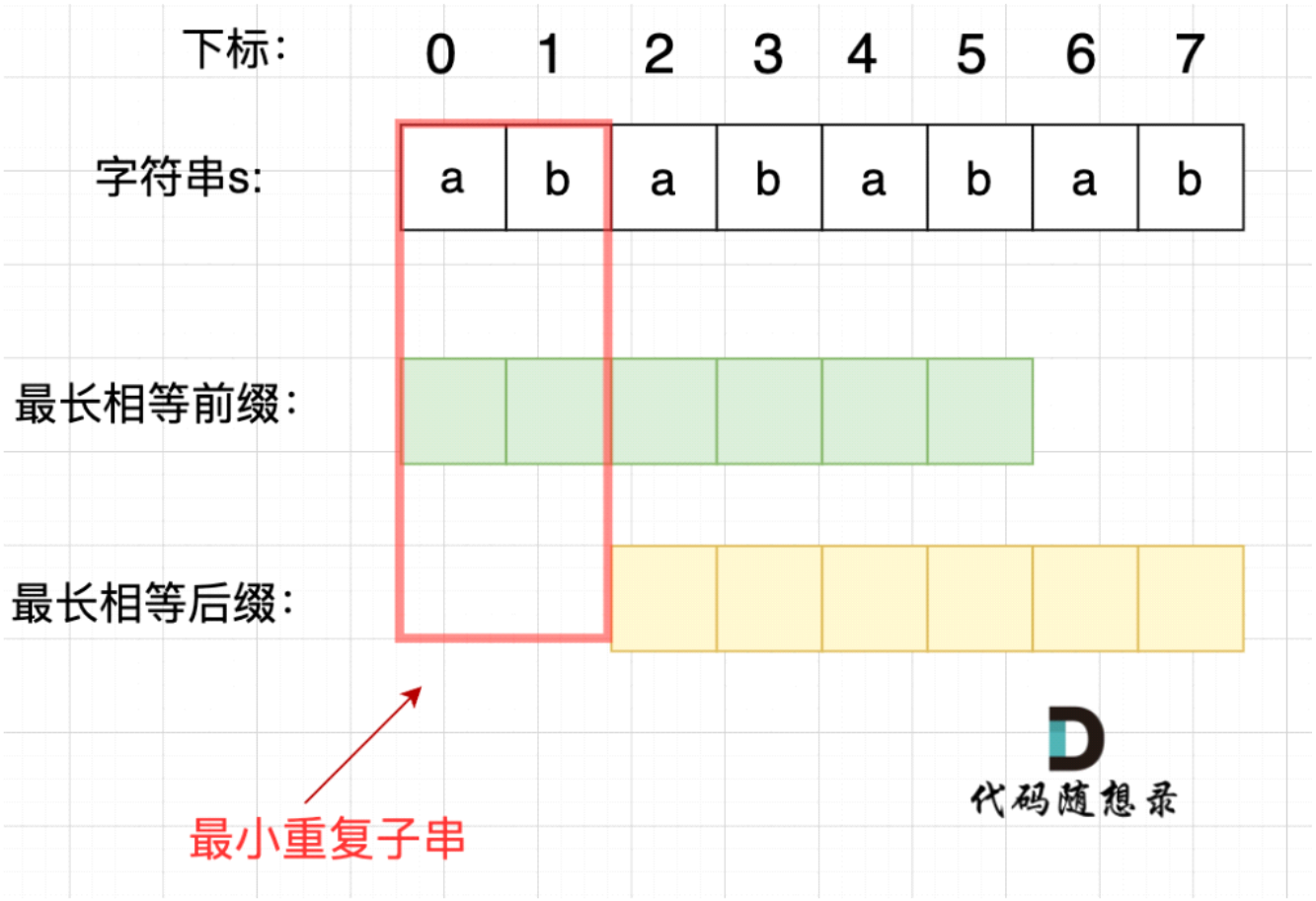
当然，我们在判断 s + s 拼接的字符串里是否出现一个s的的时候，**要刨除 s + s 的首字符和尾字符**，这样避免在s+s中搜索出原来的s，我们要搜索的是中间拼接出来的s。

```
class Solution {
public:
    bool repeatedSubstringPattern(string s) {
        string temp=s+s;
        temp.erase(temp.begin());
        temp.erase(temp.end()-1);
        return temp.find(s)!=-1;
    }
};
```

kmp算法（变形，仅使用原理而非算法本身）

核心结论：在由重复子串组成的字符串中，最长相等前后缀不包含的子串就是最小重复子串

在由重复子串组成的字符串中，最长相等前后缀不包含的子串就是最小重复子串，这里那字符串s：ababab 来举例，ab就是最小重复单位，如图所示：



若最小重复子串的长度能被最长公共前后缀的长度整除，那么该字符串由重复多次子串构成

假设字符串s使用多个重复子串构成（这个子串是最小重复单位），重复出现的子字符串长度是x，所以s是由n * x组成。

因为字符串s的最长相同前后缀的的长度一定是不包含s本身，所以 最长相同前后缀长度必然是m * x，而且 n - m = 1，（这里如果不懂，看上面的推理）

所以如果 nx % (n - m)x = 0，就可以判定有重复出现的子字符串。

next 数组记录的就是最长相同前后缀 字符串：KMP算法精讲 这里介绍了什么是前缀，什么是后缀，什么又是最长相同前后缀)， 如果 next[len - 1] != -1，则说明字符串有最长相同的前后缀（就是字符串里的前缀子串和后缀子串相同的最长长度）。

最长相等前后缀的长度为：next[len - 1] + 1。(这里的next数组是以统一减一的方式计算的，因此需要+1，两种计算next数组的具体区别看这里：字符串：KMP算法精讲)

数组长度为：len。

如果len % (len - (next[len - 1] + 1)) == 0 ，则说明数组的长度正好可以被 (数组长度-最长相等前后缀的长度) 整除 ，说明该字符串有重复的子字符串。

```
class Solution {
public:
    void getNext(int *next,const string &s){
        int j=0;
        next[0]=j;
        for(int i=1;i<s.size();i++){
            while(j>0&&s[j]!=s[i]){
                j=next[j-1];
            }
            if(s[j]==s[i]){
                j++;
            }
            next[i]=j;
        }
    }
    bool repeatedSubstringPattern(string s) {
        int next[s.size()];
        getNext(next,s);
        int len=next[s.size()-1];
        return (len%(s.size()-len)==0)&&(len!=0); //不等于0保证主串有最长公共前后缀
    }
};
```