youngyangyang04 Update History

4 contributors

301 lines (236 sloc) 9.15 KB

> KMP算法还能干这个

459.重复的子字符串

https://leetcode-cn.com/problems/repeated-substring-pattern/

给定一个非空的字符串，判断它是否可以由它的一个子串重复多次构成。给定的字符串只含有小写英文字母，并且长度不超过10000。

示例 1:
输入: "abab"
输出: True
解释: 可由子字符串 "ab" 重复两次构成。

示例 2:
输入: "aba"
输出: False

示例 3:
输入: "abcabcabcabc"
输出: True
解释: 可由子字符串 "abc" 重复四次构成。 (或者子字符串 "abcabc" 重复两次构成。)

思路

这又是一道标准的KMP的题目。

如果KMP还不够了解，可以看我的B站：

- 帮你把KMP算法学个通透！B站（理论篇）
- 帮你把KMP算法学个通透！（求next数组代码篇）

我们在字符串：KMP算法精讲里提到了，在一个串中查找是否出现过另一个串，这是KMP的看家本领。

那么寻找重复子串怎么也涉及到KMP算法了呢？

这里就要说一说next数组了，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。

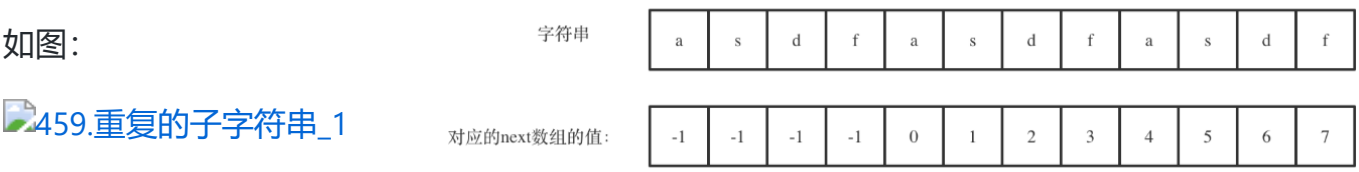
数组长度为：len。

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

Q1 **数组长度减去最长相同前后缀的长度相当于是第一个周期的长度，也就是一个周期的长度，如果这个周期可以被整除，就说明整个数组就是这个周期的循环。**

强烈建议大家把next数组打印出来，看看next数组里的规律，有助于理解KMP算法

如图：



459.重复的子字符串_1

next[len - 1] = 7，next[len - 1] + 1 = 8，8就是此时字符串asdfasdfasdf的最长相同前后缀的长度。

(len - (next[len - 1] + 1)) 也就是： 12(字符串的长度) - 8(最长公共前后缀的长度) = 4， 4正好可以被 12(字符串的长度) 整除，所以说明有重复的子字符串（asdf）。

C++代码如下：（这里使用了前缀表统一减一的实现方式）

```
class Solution {
public:
    void getNext (int* next, const string& s){
        next[0] = -1;
        int j = -1;
        for(int i = 1;i < s.size(); i++){
            while(j >= 0 && s[i] != s[j+1]) {
                j = next[j];
            }
            if(s[i] == s[j+1]) {
                j++;
            }
            next[i] = j;
        }
    }
```

```
    bool repeatedSubstringPattern (string s) {
        if (s.size() == 0) {
            return false;
        }
        int next[s.size()];
        getNext(next, s);
        int len = s.size();
        if (next[len - 1] != -1 && len % (len - (next[len - 1] + 1)) == 0) {
            return true;
        }
        return false;
    }
};
```

前缀表（不减一）的C++代码实现

```
class Solution {
public:
    void getNext (int* next, const string& s){
        next[0] = 0;
        int j = 0;
        for(int i = 1;i < s.size(); i++){
            while(j > 0 && s[i] != s[j]) {
                j = next[j - 1];
            }
            if(s[i] == s[j]) {
                j++;
            }
            next[i] = j;
        }
    }
    bool repeatedSubstringPattern (string s) {
        if (s.size() == 0) {
            return false;
        }
        int next[s.size()];
        getNext(next, s);
        int len = s.size();
        if (next[len - 1] != 0 && len % (len - (next[len - 1] )) == 0) {
            return true;
        }
        return false;
    }
};
```

拓展

在字符串：KMP算法精讲中讲解KMP算法的基础理论，给出next数组究竟是如何来了，前缀表又是怎么回事，为什么要选择前缀表。

讲解一道KMP的经典题目，力扣：28. 实现 strStr()，判断文本串里是否出现过模式串，这里涉及到构造next数组的代码实现，以及使用next数组完成模式串与文本串的匹配过程。

后来很多同学反馈说：搞不懂前后缀，什么又是最长相同前后缀（最长公共前后缀我认为这个用词不准确），以及为什么前缀表要统一减一（右移）呢，不减一行不行？针对这些问题，我在字符串：KMP算法精讲给出了详细的讲解。

其他语言版本

Java：

```
class Solution {
    public boolean repeatedSubstringPattern(String s) {
        if (s.equals("")) return false;

        int len = s.length();
        // 原串加个空格(哨兵)，使下标从1开始，这样j从0开始，也不用初始化了
        s = " " + s;
        char[] chars = s.toCharArray();
        int[] next = new int[len + 1];

        // 构造 next 数组过程，j从0开始(空格)，i从2开始
        for (int i = 2, j = 0; i <= len; i++) {
            // 匹配不成功，j回到前一位置 next 数组所对应的值
            while (j > 0 && chars[i] != chars[j + 1]) j = next[j];
            // 匹配成功，j往后移
            if (chars[i] == chars[j + 1]) j++;
            // 更新 next 数组的值
            next[i] = j;
        }

        // 最后判断是否是重复的子字符串，这里 next[len] 即代表next数组末尾的值
        if (next[len] > 0 && len % (len - next[len]) == 0) {
            return true;
        }
        return false;
    }
}
```

Python：

这里使用了前缀表统一减一的实现方式

```
class Solution:
    def repeatedSubstringPattern(self, s: str) -> bool:
        if len(s) == 0:
            return False
        nxt = [0] * len(s)
        self.getNext(nxt, s)
        if nxt[-1] != -1 and len(s) % (len(s) - (nxt[-1] + 1)) == 0:
            return True
        return False

    def getNext(self, nxt, s):
        nxt[0] = -1
        j = -1
        for i in range(1, len(s)):
```

```
            while j >= 0 and s[i] != s[j+1]:
                j = nxt[j]
            if s[i] == s[j+1]:
                j += 1
            nxt[i] = j
        return nxt
```

前缀表（不减一）的代码实现

```
class Solution:
    def repeatedSubstringPattern(self, s: str) -> bool:
        if len(s) == 0:
            return False
        nxt = [0] * len(s)
        self.getNext(nxt, s)
        if nxt[-1] != 0 and len(s) % (len(s) - nxt[-1]) == 0:
            return True
        return False

    def getNext(self, nxt, s):
        nxt[0] = 0
        j = 0
        for i in range(1, len(s)):
            while j > 0 and s[i] != s[j]:
                j = nxt[j - 1]
            if s[i] == s[j]:
                j += 1
            nxt[i] = j
        return nxt
```

Go：

这里使用了前缀表统一减一的实现方式

```
func repeatedSubstringPattern(s string) bool {
        n := len(s)
        if n == 0 {
                return false
        }
        next := make([]int, n)
        j := -1
        next[0] = j
        for i := 1; i < n; i++ {
                for j >= 0 && s[i] != s[j+1] {
                        j = next[j]
                }
                if s[i] == s[j+1] {
                        j++
                }
                next[i] = j
        }
        // next[n-1]+1 最长相同前后缀的长度
        if next[n-1] != -1 && n%(n-(next[n-1]+1)) == 0 {
                return true
        }
```

```
        return false
}
```

前缀表（不减一）的代码实现

```
func repeatedSubstringPattern(s string) bool {
        n := len(s)
        if n == 0 {
                return false
        }
        j := 0
        next := make([]int, n)
        next[0] = j
        for i := 1; i < n; i++ {
                for j > 0 && s[i] != s[j] {
                        j = next[j-1]
                }
                if s[i] == s[j] {
                        j++
                }
                next[i] = j
        }
        // next[n-1]  最长相同前后缀的长度
        if next[n-1] != 0 && n%(n-next[n-1]) == 0 {
                return true
        }
        return false
}
```

- 作者微信：程序员Carl
- B站视频：代码随想录
- 知识星球：代码随想录