入选理由

1. 字符串匹配问题经典中的经典，本次专题不要求深度，要求掌握即可
2. 一题两做，本次要求大家用 BF 和 RK 两种方法 AC

标签

- 字符串

难度

- 简单

题目地址（28 实现 strStr()-BF&RK）

https://leetcode-cn.com/problems/implement-strstr/

题目描述

```
实现 strStr() 函数。

给定一个 haystack 字符串和一个 needle 字符串，在 haystack 字符串中找出 needle 字符串出现的第一个位置 (从0开始)。如果不存在，则返回  -1。

示例 1:

输入: haystack = "hello", needle = "ll"
输出: 2
示例 2:

输入: haystack = "aaaaa", needle = "bba"
输出: -1
说明:

当 needle 是空字符串时，我们应当返回什么值呢？这是一个在面试中很好的问题。

对于本题而言，当 needle 是空字符串时我们应当返回 0 。这与C语言的 strstr() 以及 Java的 indexOf() 定义相符。
```

前置知识

- 滑动窗口
- 字符串
- Hash 运算

暴力法

思路

该题基本上就是字符串匹配问题的入门，选这个题的原因也很简单，一般 KMP&RK 算法出现在面试中的频率相对较低，因此不需要过分考察深度，只需要掌握该算法基本即可。该题稍微注意一下的地方就是待匹配串可能多个符合模式串的子串，我们只需要返回第一次匹配成功的位置即可。

代码

代码支持： Python3, Java, CPP

Python3 Code：

```python
class Solution:
    def strStr(self, haystack, needle):
        """
        :type haystack: str
        :type needle: str
        :rtype: int
        """
        lenA, lenB = len(haystack), len(needle)
        if not lenB:
            return 0
        if lenB > lenA:
            return -1

        for i in range(lenA - lenB + 1):
            if haystack[i:i + lenB] == needle:
                return i
        return -1
```

Java Code:

```java
class Solution {
    public int strStr(String haystack, String needle) {
        if (needle == null || needle.isEmpty()) {
```

```
            return 0;
        }

        int m = haystack.length();
        int n = needle.length();
        if (m < n) {
            return -1;
        }

        for (int i=0; i<=m-n; i++) {
            String subStr = haystack.substring(i, i + n);
            if (subStr.equals(needle)) {
                return i;
            }
        }

        return -1;
    }
}
```

CPP Code:

```
class Solution {
public:
    int strStr(string haystack, string needle) {
        if (haystack.length() < needle.length()) {
            return -1;
        }
        else {
            for (int i=0; i<haystack.length()-needle.length()+1; i++) {
            if (haystack.substr(i, needle.length()) == needle) return i;
        }
        return -1;
        }
    }
};
```

复杂度分析

设：待匹配串长为N，模式串串长为M

- 时间复杂度: $O(N * M)$
- 空间复杂度：$O(1)$

RK（滚动哈希）

思路

这个算法也有一个比较形象的名字（滚动哈希）。

首先我们把 needle 用 hash 算法计算一次哈希值，接下来我们的目标就是在 haystack 中找到一个连续子串使得其哈希值也是 needle 计算出来的哈希值。

由于 haystack 和 needle 最多只有 26 个字符，因此我们可以用 26 进制进制编码。

比如 hi 就编码为 26 * (ord('h') - ord('a')) + ord('i') - ord('a')， 这就是我们的哈希算法。

于是这个问题可以使用滑动窗口来解决。具体来说：

- 先计算出和 needle 长度一致的哈希值，也就是 haystack[:len(needle)] 的哈希值。
- 然后我们需要移动窗口，每次移动一格。 这样哈希值仅需要减去左侧移除窗口的值和右侧移入窗口的值即可。
- 滚动过程发现哈希值和 needle 的哈希值一致，说明其可能是一个潜在的答案（存在碰撞的可能，即两个不同字符串哈希一样），我们再最终检测一次二者是否相等。如果相等则说明找到了，直接返回。
- 到最后都没有找到，则返回 -1。

计算哈希值通常的做法就是上面的编码方案，再加上模一个素数。

这个素数对算法正确性没有任何影响，但是对算法效率影响很大。因此

1. 两个哈希一样，我们还会去最终 check 一次。
2. 这个哈希算法如果判断特别厉害。最坏情况所有的字符串哈希出来都是一样的，这个时候和朴素的暴力解法时间复杂度相同。也就是说哈希算法选不好，还不如暴力。这也是这个算法的最大弊端。

代码

代码支持： Java, Python3

Java Code:

```java
class Solution {

    int prime  = 13; // 一个素数
    int d = 26;

    public int strStr(String haystack, String needle) {


        if(needle.equals("")) return 0;
        if(haystack.equals("") || haystack.length() < needle.length()) return -1;

        int n = haystack.length();
        int m = needle.length();
```

```
        long pHashVal = initHash(needle , m);
        long tHashVal = initHash(haystack , m);

        for(int i = 0 ; i <= n-m ; i++){
            if(i > 0 && i <= n - m ){
                tHashVal = reacalHash(haystack , i-1 , i + m -1 , tHashVal , m);
            }
            if(pHashVal == tHashVal && isEquals(haystack , needle , i)){
                return i;
            }
        }
        return -1;

    }

    public long initHash(String text , int end){
        long hashval = 0;

        for(int i = 0 ; i < end ; i++){
            hashval += (text.charAt(i) - 'a')* Math.pow(d , end-i-1);
        }
        hashval %= prime;
        return hashval;
    }

    public long reacalHash(String text , int OldIndex , int newIndex ,long hashval , int patternLength){
        long newHash = (hashval -  (text.charAt(OldIndex) - 'a') * ((long)(Math.pow(d , patternLength-1)) % prime) ) * d
        return  newHash % prime;
    }

    public boolean isEquals(String text , String pattern , int tStart){
        int end = tStart + pattern.length();
        int pStart = 0;

        while(tStart < end){
            if((text.charAt(tStart) - 'a') != (pattern.charAt(pStart) - 'a')){
                return false;
            }

            tStart++;
            pStart++;
        }
        return true;
    }
}
```

代码稍作解释一下，我这用个 101 的素数太小了，但是因为毕竟在刷题，不用设置过大，如果工程上使用还是要谨慎选取的。

Python3 Code:

```python
class Solution:
    def strStr(self, haystack: str, needle: str) -> int:
        if not haystack and not needle:
            return 0
        if not haystack or len(haystack) < len(needle):
            return -1
        if not needle:
            return 0
        hash_val = 0
        target = 0
        prime = 101
        for i in range(len(haystack)):
            if i < len(needle):
                hash_val = hash_val * 26 + (ord(haystack[i]) - ord("a"))
                hash_val %= prime
                target = target * 26 + (ord(needle[i]) - ord("a"))
                target %= prime
            else:
                hash_val = (
                    hash_val - (ord(haystack[i - len(needle)]) - ord("a")) * ((26 ** (len(needle) - 1) )% prime)
                ) * 26 + (ord(haystack[i]) - ord("a"))
                hash_val %= prime
            if i >= len(needle) - 1 and hash_val == target and haystack[i-len(needle)+1:i+1] == needle:
                return i - len(needle) + 1
        return 0 if hash_val == target and haystack[i-len(needle)+1:i+1] == needle else -1
```

复杂度分析

设：待匹配串长为N，模式串串长为M

- 时间复杂度：一般情况是$\mathrm{O}(\mathrm{N}+\mathrm{M})$，最坏和暴力法一样，为 $\mathrm{O}(\mathrm{M} * \mathrm{N})$。
- 空间复杂度：$\mathrm{O}(1)$

进阶

- 能否实现查找所有匹配成功的位置?

上一页 下一页