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 KMP 方法 AC

标签

- 字符串

难度

- 简单

题目地址（28 实现 strStr()-KMP）

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

分析

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。

进阶：能否实现查找所有匹配成功的位置?

代码

代码支持：Java，Python3，JS，CPP

Java Code：

```java
class Solution {
    public int strStr(String haystack, String needle) {

        if (needle.length() == 0)
            return 0;

        int i = 0, j = 0;

        int[] next = getNext(needle);

        while (i < haystack.length() && j < needle.length()) {

            if (haystack.charAt(i) == needle.charAt(j)) {

                i++;
                j++;
            } else {

                if (j > 0)
                    j = next[j - 1];
                else
                    i++;
            }

            if (j == needle.length())
                return i - j;
        }
```

```
        return -1;
    }

    public int[] getNext(String pattern) {

        int[] next = new int[pattern.length()];

        int j = 0;
        for (int i = 1; i < pattern.length(); i++) {

            if (pattern.charAt(i) == pattern.charAt(j))
                next[i] = ++j;
            else {

                while (j > 0 && pattern.charAt(j) != pattern.charAt(i))
                    j = next[j - 1];

                if (pattern.charAt(i) == pattern.charAt(j))
                    next[i] = ++j;
            }
        }

        return next;
    }
}
```

Python3 Code:

```
class Solution:
    def strStr(self, haystack: str, needle: str) -> int:
        n, m = len(haystack), len(needle)
        if not needle: return 0
        if m > n: return -1

        # 维护一个pattern的next数组
        def KMPNext(needle):
            next = [None] * len(needle)
            j = 0
            for i in range(1, len(needle)):
                while needle[i] != needle[j]:
                    if j > 0:
                        j = next[j - 1]
                    else:
                        next[i] = 0
                        break
                if needle[i] == needle[j]:
                    j += 1
                    next[i] = j
            return next
```

```
        next = KMPNext(needle)
        i, j = 0, 0

        while i < n and j < m:
            if haystack[i] == needle[j]:
                i += 1
                j += 1
            else:
                if j > 0:
                    j = next[j - 1]
                else:
                    i += 1
            if j == m:
                return i - j

        return -1
```

JS Code：

```
var strStr = function (haystack, needle) {
  if (needle.length === 0) return 0;

  const n = haystack.length,
    m = needle.length;
  const s = " " + haystack;
  const p = " " + needle;
  const next = new Array(m + 1).fill(0);

  for (let i = 2, j = 0; i <= m; i++) {
    while (j > 0 && p[i] != p[j + 1]) j = next[j];
    if (p[i] == p[j + 1]) j++;
    next[i] = j;
  }

  for (let i = 1, j = 0; i <= n; i++) {
    while (j > 0 && s[i] != p[j + 1]) j = next[j];
    if (s[i] == p[j + 1]) j++;
    if (j == m) return i - m;
  }
  return -1;
};
```

CPP Code:

```
class Solution {
public:
```

```
    int strStr(string haystack, string needle) {
        int n = haystack.size(), m = needle.size();
        if (m == 0) {
            return 0;
        }
        vector<int> pi(m);
        for (int i = 1, j = 0; i < m; i++) {
            while (j > 0 && needle[i] != needle[j]) {
                j = pi[j - 1];
            }
            if (needle[i] == needle[j]) {
                j++;
            }
            pi[i] = j;
        }
        for (int i = 0, j = 0; i < n; i++) {
            while (j > 0 && haystack[i] != needle[j]) {
                j = pi[j - 1];
            }
            if (haystack[i] == needle[j]) {
                j++;
            }
            if (j == m) {
                return i - m + 1;
            }
        }
        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：

- BF：$\mathrm{O}(\mathrm{NM})$
- RK：若 hash function 选的差，冲突多，则最坏是(NM)，一般情况是$\mathrm{O}(\mathrm{N}+\mathrm{M})$
- KMP：$\mathrm{O}(\mathrm{N}+\mathrm{M})$

空间复杂度：

- BF：$\mathrm{O}(1)$
- RK：$\mathrm{O}(1)$
- KMP：$\mathrm{O}(\mathrm{M})$

上一页 下一页