链滴

字符串匹配算法之 Kmp 算法

作者：zyjImmortal
原文链接：https://ld246.com/article/1579143327505
来源网站：链滴

KMP算法核心思路

kmp算法也是一种字符串匹配算法，但是他和BM算法不同的是，从前往后开始匹配。共同点是都是找出滑动最大位数的规律。

假设现在有两个字符串，一个是abaabaabbabaaabaabbabaab,起个名叫m，另一个abaabbabaab p，

abaabaabbabaaabaabbabaab

abaabbabaab

两个字符串从头开始匹配的，匹配到第六个字符的时候出现不匹配的情况，这个时候把主串的第六位符叫做坏字符，前面匹配到的叫做好前缀，这两个称呼都是针对主串的。当出现不匹配的坏字符的时，模式串滑动多少位，能够更快的匹配完?

KMP算法就是在视图寻找一种规律，在模式传和主串匹配的过程中，当遇到坏字符后，能否找到一种律，将模式串一次性滑动很多位，而且匹配的字符尽可能的不再匹配。

观察主串的好前缀其实也是模式串的前缀子串，我们只要找出公共的子串，公共子串同时也是后缀子，就可以确定滑动的位数，以上面的两个字符串为例，最长的公共子串是ab，长度为2，坏字符的索是5，将模式串滑动到公共子串的位置所需滑动的位数5-2=3，如下图：

abaabaabbabaaabaabbabaab

abaabbabaab

这个时候需要重新，模式串中坏字符对应指针的位置，这个新位置是在好前缀公共子串的后面，也就是公共子串最后一位字符的索引+1.

假设坏字符对应主串的位置是i，在模式串的位置是j，最长公共子串，就是模式串p[0]到p[j]之间子串，前缀和后缀的最长公共子串。那我们是不是可以对每个位置的模式串子串求最长公共子串，然后把存到数组里，当主串和模式串匹配到坏字符的时候，直接从数组中取出最长公共子串，然后就可以得滑动的位数了。

那我们来看下一下这个数组是如何构造的以及都有什么特征。

数组的值存储的是，最长可匹配前缀子串的结尾字符的下标，这里我们前缀子串和后缀子串去最长数的下标是每个前缀子串的下标

模式字符串 a b a b a c d

模式串前缀	**前缀子串结尾字符下标**	**最长可匹配前子串结尾字符下标 next值**
a	0	-1(表示不存在) next[] = -1
a b	1	-1 next[1] = -1
a b a	2	0 next[2] = 0
a b a b	3	1 next[3] = 1
a b a b a	4	2 next[4] = 2
a b a b a c	5	-1 next[5] -1

代码实现

```
public class Kmp {
    /**
     * next的数组推到过程是假设已经有部分数据计算出来，以此为基础计算后面的
     * 这里说的最公共子串，指的是，模式串从开头到当前位置的子串的所有前缀子串和后缀子串的最公共子串
     * <p>
     * 假设模式串中0-9位对应的公共最长子串已经计算出来，此时要计算第10位字符'a'对应的最大公子串长度
     * 首先考虑加了一个字符，最大公共子串长度是否会加一，这个时候可以利用已经求出的第9位的大公共子串，
     * 假设第九位最大公共子串是abaa，此时看这个前缀子串的后面一个字符是否和新字符'a'相等，果相等，那最长公共子串长度就+1
     * <p>
     * 如果不相等，就只能考虑最长长度不变或者减小的情况了。
     * 此时要找的是，最大前缀的前缀和最大后缀加'a'字符组合的后缀的公共最长的子串了，
     * <p>
     * 最长前缀和最长后缀是一样的，那么问题就转换成，最长前缀加新字符的公共最长的子串了，就回到了开始时的规则了
     * <p>
     * 整体解决思路类似于一个动态规划的问题，求某个位置的所有前缀子串和所有后缀子串的公共子长度，
     * 可以通过前一个位置的公最长共子串长度得出。
     *
```

```
     * @param b
     * @param m
     * @return
     */
    public static int[] getNexts(char[] b, int m) {
        int[] next = new int[m];
        next[0] = -1;
        int j = -1;
        for (int i = 1; i < m; i++) {
            while (b[i] != b[j + 1] && j >= 0) {
                j = next[j];
            }
            // 如果i位置的字符和最大前缀子串的后面一个字符相等，那么i位置的最大前缀子串长度就+1
            if (b[i] == b[j + 1]) {
                j++;
            }
            next[i] = j;
        }
        return next;
    }

    /**
     * next 数组存的是当前
     * 从模式串头部到当前位置的这一子串的，
     * 所有前缀子串和所有后缀子串匹配到最长公共子串时
     * 前缀子串的最后一个字符的索引
     *
     * @param a
     * @param n
     * @param b
     * @param m
     * @return
     */
    public static int kmp(char[] a, int n, char[] b, int m) {
        int[] next = getNexts(b, m);
        // i 是主串指针，j是模式串指针
        int j = 0;
        for (int i = 0; i < n; i++) {
            while (j > 0 && b[j] != a[i]) {
                j = next[j - 1] + 1;
            }
            if (a[i] == b[j]) {
                j++;
            }

            if (j == m) {
                return i - m + 1;
            }
        }
        return -1;
    }

}
```

实力有限，如果看不懂我说的呢，就给大家推荐知乎上的一个问答，里面说的很是通俗易懂如何更好理解和掌握 KMP 算法?前三个回答强烈建议大家细读细品。

原文链接：字符串匹配算法之 Kmp 算法