

KMP算法

一、介绍

KMP算法，又称作“看猫片”算法（误），是一种改进的字符串模式匹配算法，可以在 $O(n+m)$ 的时间复杂度以内完成字符串的匹配操作，其核心思想在于：当一趟匹配过程中出现字符不匹配时，不需要回溯主串的指针，而是利用已经得到的“部分匹配”，将模式串尽可能多地向右“滑动”一段距离，然后继续比较。

二、最佳应用--字符串匹配问题

1) 有一个字符串 `str1= "BBCABCDABABCDABCDABDE"`，和一个子串 `str2="ABCDABD"`

2) 现在要判断 `str1` 是否含有 `str2`，如果存在，就返回第一次出现的位置，如果没有，则返回-1

三、思路分析图解

首先理解几个概念：

1、前缀，后缀

字符串：**"bread"**

前缀：**b , br , bre , brea**

后缀：**read , ead , ad , d**

“部分匹配值”就是“前缀”和“后缀”的最长的共有元素的长度。

以“ABCDABD”为例，

- “A” 的前缀和后缀都为空集，共有元素的长度为0；
- “AB” 的前缀为[A]，后缀为[B]，共有元素的长度为0；
- “ABC” 的前缀为[A, AB]，后缀为[BC, C]，共有元素的长度为0；
- “ABCD” 的前缀为[A, AB, ABC]，后缀为[BCD, CD, D]，共有元素的长度为0；
- “ABCD A” 的前缀为[A, AB, ABC, ABCD]，后缀为[BCDA, CDA, DA, A]，共有元素为“A”，长度为1；
- “ABCDAB” 的前缀为[A, AB, ABC, ABCD, ABCDA]，后缀为[BCDAB, CDAB, DAB, AB, B]，共有元素为“AB”，长度为2；
- “ABCDABD” 的前缀为[A, AB, ABC, ABCD, ABCDA, ABCDAB]，后缀为[BCDABD, CDABD, DABD, ABD, BD, D]，共有元素的长度为0。

2、“部分匹配”

实质是，有时候，字符串头部和尾部会有重复。比如，“ABCDAB”之中有两个“AB”，那么它的“部分匹配值”就是2（“AB”的长度）。搜索词移动的时候，第一个“AB”向后移动4位（字符串长度-部分匹配值），就可以来到第二个“AB”的位置。

搜索词	A	B	C	D	A	B	D
部分匹配值	0	0	0	0	1	2	0

3、思路分析

举例来说，有一个字符串 Str1 = “BBC ABCDAB ABCDABCDABDE”，判断，里面是否包含另一个字符串 Str2 = “ABCDABD”？

1. 首先，用Str1的第一个字符和Str2的第一个字符去比较，不符合，关键词向后移动一位

BBC ABCDAB ABCDABCDABDE
ABCDABD

2. 重复第一步，还是不符合，再后移

BBC ABCDAB ABCDABCDABDE
ABCDABD

3. 一直重复，直到Str1有一个字符与Str2的第一个字符符合为止

BBC ABCDAB ABCDABCDABDE
ABCDABD

4. 接着比较字符串和搜索词的下一个字符，还是符合。

BBC ABCDAB ABCDABCDABDE
ABCDABD

5. 遇到Str1有一个字符与Str2对应的字符不符合。

BBC ABCDAB ABCDABCDABDE
ABCDABD

6. 这时候，想到的是继续遍历Str1的下一个字符，重复第1步。（其实是很不明智的，因为此时BCD已经比较过了，没有必要再做重复的工作，一个基本事实是，当空格与D不匹配时，你其实知道前面六个字符是” ABCDAB”。KMP 算法的想法是，设法利用这个已知信息，不要把” 搜索位置” 移回已经比较过的位置，继续把它向后移，这样就提高了效率。）

BBC ABCDAB ABCDABCDABDE
ABCDABD

7. 怎么做到把刚刚重复的步骤省略掉？可以对Str2计算出一张《部分匹配表》，这张表的产生在后面介绍

搜索词	A	B	C	D	A	B	D
部分匹配值	0	0	0	0	1	2	0

8. 已知空格与D不匹配时，前面六个字符” ABCDAB” 是匹配的。查表可知，最后一个匹配字符B对应的” 部分匹配值” 为2，因此按照下面的公式算出向后移动的位数：

移动位数 = 已匹配的字符数 - 对应的部分匹配值

因为 6 - 2 等于4，所以将搜索词向后移动 4 位。

9. 因为空格与C不匹配，搜索词还要继续往后移。这时，已匹配的字符数为 2（” AB”），对应的” 部分匹配值” 为0。所以，移动位数 = 2 - 0，结果为 2，于是将搜索词向后移 2 位。

BBC ABCDAB ABCDABCDABDE
ABCDABD

10. 因为空格与A不匹配，继续后移一位。

BBC ABCDAB ABCDABCDABDE
ABCDABD

11. 逐位比较，直到发现C与D不匹配。于是，移动位数 = $6 - 2$ ，继续将搜索词向后移动 4 位。

BBC ABCDAB ABCDABCDABDE
ABCDABD

12. 逐位比较，直到搜索词的最后一位，发现完全匹配，于是搜索完成。如果还要继续搜索（即找出全部匹配），移动位数 = $7 - 0$ ，再将搜索词向后移动 7 位，这里就不再重复了。

BBC ABCDAB ABCDABCDABDE
ABCDABD

到此KMP算法思想分析完毕！

四、代码实现

kmp匹配算法

```
//写出我们的kmp搜索算法
/**
 *
 * @param str1 源字符串
 * @param str2 子串
 * @param next 部分匹配表, 是子串对应的部分匹配表
 * @return 如果是-1就是没有匹配到, 否则返回第一个匹配的位置
 */
public static int kmpSearch(String str1, String str2, int[] next) {

    //遍历
    for(int i = 0, j = 0; i < str1.length(); i++) {

        //需要处理 str1.charAt(i) != str2.charAt(j), 去调整j的大小
        //KMP算法核心点
        //直接返回到共有元素后面, 判断之后的字符串是否相等, 即第11步
        while( j > 0 && str1.charAt(i) != str2.charAt(j)) {
            j = next[j-1]; //返回到共有元素部分
        }

        if(str1.charAt(i) == str2.charAt(j)) {
            j++;
        }
        if(j == str2.length()) { //找到了 // j = 3 i
            return i - j + 1;
        }
    }
    return -1;
}
```

部分匹配表算法

```
//获取到一个字符串(子串) 的部分匹配值表
public static int[] kmpNext(String dest) {
    //创建一个next 数组保存部分匹配值
    int[] next = new int[dest.length()];
    next[0] = 0; //如果字符串是长度为1 部分匹配值就是0
    for(int i = 1, j = 0; i < dest.length(); i++) {
```

```
//当dest.charAt(i) != dest.charAt(j) , 我们需要从next[j-1]获取新的j
//直到我们发现 有 dest.charAt(i) == dest.charAt(j)成立才退出
//这时kmp算法的核心点
while(j > 0 && dest.charAt(i) != dest.charAt(j)) {
    j = next[j-1];
}

//当dest.charAt(i) == dest.charAt(j) 满足时, 部分匹配值就是+1
if(dest.charAt(i) == dest.charAt(j)) {
    j++;
}
next[i] = j;
}
return next;
}
```