字符串模式匹配算法

1. KMP算法

术语：

源串： 字符串数据源

模式串：待定位的字符串

该算法的核心思路：

**源串的位置保持不动，通过移动模式串相对源串的位置来进行匹配。**

接下来要解决的问题是移动多少位，如何移动？

解决这个问题需要先来对模式串进行分析，先引入几个关键的概念

**字符串前缀：**一个字符串所能包含的所有前缀，如”ABC”的前缀：”A”,”AB”

**字符串后缀：**一个字符串所能包含的所有后缀，如”ABC”的后缀：”C”,”BC”

**字符串前缀后缀的公共元素的最大长度：**字符串前缀和后缀包含的最大的公共元素的长度

例如：模式串“adgadsc”的各个子串的前缀后缀以及公共元素最大长度如下表所示：

|  |  |  |  |
| --- | --- | --- | --- |
| 模式串的子串 | 前缀 | 后缀 | 最大公共元素长度 |
| a | 空 | 空 | 0 |
| ad | a | d | 0 |
| adg | a,ad | g,dg | 0 |
| adga | **a**,ad,adg | **a**,ga,dga | 1 |
| adgad | a,**ad**,adg,adga | d,**ad**,gad,dgad | 2 |
| adgads | a,ad,adg,adga,adgad | s,ds,ads,gads,dgads | 0 |
| adgadsc | a,ad,adg,adga,adgad,adgads | c,sc,dsc,adsc,gadsc,dgadsc | 0 |

当匹配失败的时候，**模式串向右移动的位数 = 已匹配的字符数 – 失配字符的上一位字符对应的最大长度值。**

可以得到结论：当模式串的相似度越高，右移动的位数越少

**next数组：**size为模式串的长度，next[j]代表的是模式串中除索引为j的字符所构成的子串（即索引为0到j-1构成的子字符串）的前缀后缀公共元素的长度。

**next[0]特殊，需要设置next[0] = -1，这么设置的原因是：**在后面求next数组的时候需要把这个当做结束条件。

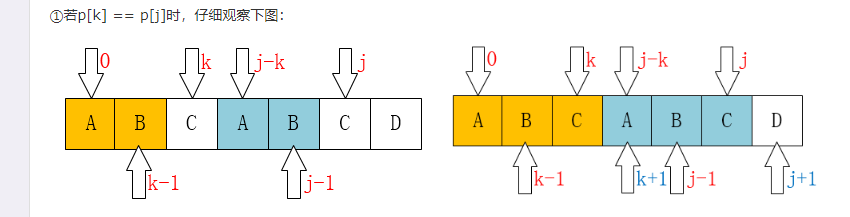
接下来需要解决的一个问题是：

已知next[0],…,next[j]的值，如何求出next[j+1]的值？

这里分为两种情况

k = next[j]

第一种情况：当char[k] = char[j+1]，则 next[j + 1] = next[j] + 1;

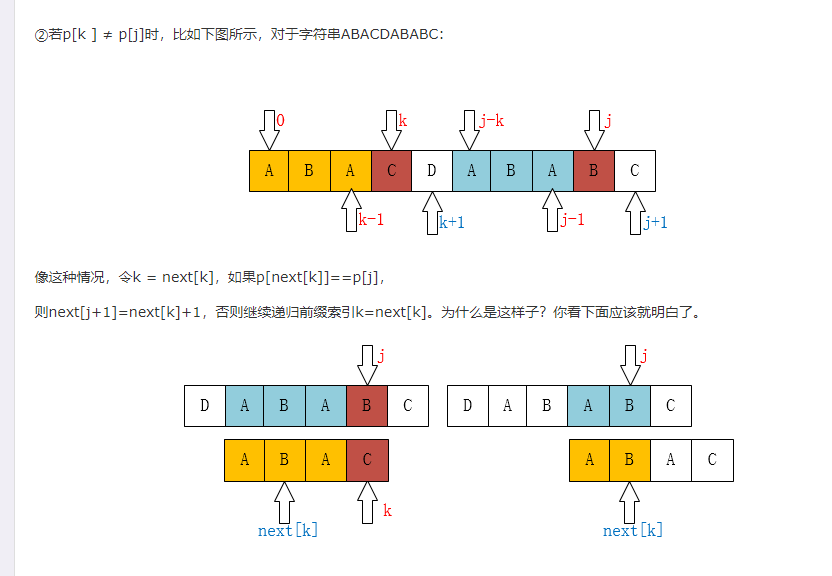


第二种情况：当char[k] != char[j + 1]时，

令lk = next[k]

当char[lk] = char[j+1]，则next[j + 1] = next[k] + 1;

否则，重复上述步骤，直到匹配或者next[lk] = -1;



**为何递归前缀索引k = next[k],就能找到长度更短的前缀后缀？**

因为这个查找是在前缀后缀公共子串中查找。

KMP算法实现类见：**KMPPatternMatch.java**