**5.3 子字符串查找**

**5.3.1 历史简介**

**5.3.2 暴力子字符串查找算法**

编程：暴力子字符串查找

在最坏情况下，暴力子字符串查找算法在长度为N的文本中查找长度为M的模式需要~NM次字符比较。

**5.3.3 Knuth-Morris-Pratt子字符串查找算法**

KMP算法的主要思想是提前判断如何重新开始查找，而这种判断只取决于模式本身。

在KMP子字符串查找算法中，不会回退文本指针i，而是使用一个数组dfa[][]来记录匹配失败时指针j应该回退多远。

对于每个字符c，在比较了c和pat.charAt(j)之后，dfa[c][j]表示的是应该和下个文本字符比较的模式字符的位置。

编程：KMP字符串查找算法

对于长度为M的模式字符串和长度为N的文本，KMP字符串查找算法访问的字符不会超过M+N个。

**5.3.4 Boyer-Moore字符串查找算法**

当可以在文本字符串中回退时，如果可以从右向左扫描模式字符串被将它和文本匹配，那么就能得到一种非常快的字符串查找算法。

使用数组right[]记录字母表中的每个字符在模式中出现的最靠右的地方。

用一个索引i在文本中从左向右移动，用另一个索引j在模式中从右向左移动。

如果M-1到0的所有j，txt.charAt(i+j)都和pat.charAt(j)相等，那么就找到了一个匹配。

匹配失败有三种情况：

1. 如果造成匹配失败的字符不包含在模式字符串中，将模式字符串向右移动j+1个位置。
2. 如果造成匹配失败的字符包含在模式字符串中，那就可以使用right[]数组来将模式字符串和文本对齐，是的该字符和它在模式字符串中出现的最右位置相匹配。
3. 如果这种方式无法增大i，那就直接将i加1来保证模式字符串至少向右移动了一个位置。

使用-1表示right[]数组中相应字符不包含在模式字符串中，这个约定能够将前两种情况合并。（将i增大j-right[txt.charAt(i+j)]）。

在一般情况下，对于长度为N的文本和长度为M的模式字符串，使用了Boyer-Moore的子字符串查找算法通过启发式处理不匹配的字符需要~N/M次字符比较。

编程：Boyer-Moore字符串匹配算法

**5.3.5 Rabin-Karp指纹字符串查找算法**

我们需要计算模式字符串的散列函数，然后用相同的散列函数计算文本中所有可能的M个字符的子字符串散列值并寻找匹配。

根据这段描述直接实现的算法将会比暴力子字符串查找算法慢很多。

Rabin和Karp发明了一种能够在常数时间内算出M个字符的子字符串散列值的方法（需要预处理）。

取余操作的一个基本性质是如果在每次算术操作之后都将结果除以Q并取余，这等价于在完成了所有算术操作之后再将最后的结果除以Q并取余。

编程：Rabin-Karp指纹字符串查找算法

**5.3.6 总结**

参考文档：各种字符串查找算法的实现的成本总结