KMP算法

2022年9月2日 10:02

next数组，即前缀表（prefix table）
用来回退，它记录了模式串与主串不匹配的时候，模式串应该从哪里开始重新匹配

前缀表：记录下标i之前（包括i）的字符串中，有多大长度的相同前缀后缀

如何利用前缀表：
当字符不匹配的时候，指针应该移动的位置是?
找到不匹配的字符时，我们查看它前一个字符的前缀表数值是多少
为什么是前一个字符的前缀表？因为要找前面字符串的最长相同的前缀和后缀

注意，前缀表统一减一，相当于指向模式串的指针左移了一位
体现在模式串与主串比较的时候，一个比较的是i和j+1（减一）、另一个是i和j（不减一）
构建next数组（减一）

```
void getNext(int* next, const string& s) {
    int j = -1;
    next[0] = j;
    for(int i = 1; i < s.size(); i++) { // 注意i从1开始
        while (j >= 0 && s[i] != s[j + 1]) { // 前后缀不相同了
            j = next[j]; // 向前回退
        }
        if (s[i] == s[j + 1]) { // 找到相同的前后缀
            j++;
        }
        next[i] = j; // 将j（前缀的长度）赋给next[i]
    }
}
```

构建next数组（不减一）

```
void getNext(int* next, const string& s) {
    int j = 0;
    next[0] = 0;
    for(int i = 1; i < s.size(); i++) {
        while (j > 0 && s[i] != s[j]) { // j要保证大于0，因为下面有取j-1作为数组下标的操作
            j = next[j - 1]; // 注意这里，是要找前一位的对应的回退位置了
        }
        if (s[i] == s[j]) {
            j++;
        }
        next[i] = j;
    }
}
```

模式匹配（next减一）

```
int j = -1; // 因为next数组里记录的起始位置为-1
for (int i = 0; i < s.size(); i++) { // 注意i就从0开始
    while(j >= 0 && s[i] != t[j + 1]) { // 不匹配
        j = next[j]; // j 寻找之前匹配的位置
    }
    if (s[i] == t[j + 1]) { // 匹配，j和i同时向后移动
        j++; // i的增加在for循环里
    }
    if (j == (t.size() - 1) ) { // 文本串s里出现了模式串t
        return (i - t.size() + 1);
    }
}
```

模式匹配（next不减一）

```
int j=0;
for(int i=0;i<s.size();i++){
    while(j>0&&s[i]!=t[j]){ //不匹配
        j=next[j-1];   // j 寻找之前匹配的位置
    }
    if(s[i]==t[j]){
        j++;
    }
    if(j==(t.size()-1)){
        return (i-t.size()+1);
    }
}
```