KMP算法整理

众所周知，字符串匹配暴力方法时间复杂度过大。经典的看毛片算法(KMP)算法，使用预处理的手段和后缀前缀特征以及递归思想，可以大幅度优化字符串匹配时间复杂度。

KMP算法思路

1. KMP就是每回模式串移动的不是一个单位移动，而是将前面匹配的都移动使得首部对应被匹配的字符串对应位置。也即是模式串得移动等同模式串移动块大小的位置。
2. 模式串移动块即字符串后缀与字符串前缀的最大共同部分。
3. 利用递归的思路预处理求解模式串移动块 从第一个位置开始循环 ，标记为q，当k位置的元素不等于q位置的元素，k递归为next[k-1],相等时k后移，使得next[q]=k;
4. 跟据next数组进行移动模式串，递归求解。

预处理代码如下

```
1.   void makeNext(const char P[],int next[])
2.   {
3.       int q,k; //声明变量
4.       int m = strlen(P);
5.       next[0] = 0;
6.       for (q = 1,k = 0; q < m; ++q)
7.       {
8.           //while语句 递归求解
9.           while(k > 0 && P[q] != P[k])
10.              k = next[k-1];
11.           //相等后移
12.          if (P[q] == P[k])
13.          {
14.              k++;
15.          }
16.          //赋值
17.          next[q] = k;
18.      }
19.  }
```

KMP核心代码

```
1.
```

```
int kmp(const char T[],const char P[],int next[])
{
    int n,m;
    int i,q;
    n = strlen(T);
    m = strlen(P);
    makeNext(P,next);
    for (i = 0,q = 0; i < n; ++i)
    {
        while(q > 0 && P[q] != T[i])
            q = next[q-1];
        if (P[q] == T[i])
        {
            q++;
        }
        if (q == m)
        {
            printf("Pattern occurs with shift:%d\n",(i-m+1));
        }
    }
}
```