

KMP

Idea: `fail[]` 失配数组的应用充分利用了模式串本身的信息减少不必要的匹配，使复杂度降至线性。具体来说，`fail[i]` 指向模式串 `t[1...i-1]` 中最长公共前后缀的前缀的下一位，如此 `i` 匹配失败时跳至 `fail[i]` 就可以继续匹配。

注意，第 `t[1...i]` 的最长公共前后缀的长度为 `fail[i+1]-1`。

Complexity: $O(n + m)$ ，其中 n 和 m 分别时目标串和模式串的长度。

Code:

```
1  int fail[N];
2  void getFail(char t[], int lent){
3      int i = 1, j = 0;
4      fail[1] = 0;
5      while(i <= lent){
6          if(!j || t[i] == t[j]) fail[++i] = ++j;
7          else j = fail[j];
8      }
9  }
10 void KMP(char s[], int lens, char t[], int lent){
11     int i = 1, j = 1;
12     while(i <= lens){
13         if(!j || s[i] == t[j]){
14             i++, j++;
15             if(j == lent + 1){
16                 printf("%d\n", i - lent);
17                 j = fail[j];
18             }
19         }
20         else j = fail[j];
21     }
22 }
```

Code (优化 `fail[]` 数组) :

```
1  int fail[N];
2  void getFail(char t[], int lent){
3      int i = 1, j = 0;
4      fail[1] = 0;
5      while(i <= lent){
6          if(!j || t[i] == t[j]){
7              i++, j++;
8              if(t[i] != t[j]) fail[i] = j;
9              else fail[i] = fail[j];
10         }
11         else j = fail[j];
12     }
13 }
14 void KMP(char s[], int lens, char t[], int lent){
15     int i = 1, j = 1;
16     while(i <= lens){
17         if(!j || s[i] == t[j]){
18             i++, j++;
19             if(j == lent + 1){
20                 printf("%d\n", i - lent);
21                 j = fail[j];
22             }
23         }
24         else j = fail[j];
25     }
```

