算法分析与设计C6-I

22373340 詹佳博

题目描述

妮妮面前有一个巨大的宝箱，宝箱上有一个锁，锁上有一串仅包含**小写字母**的字符串 s。妮妮在寻宝的过程中获得了 m 个不同的钥匙，每个钥匙上也有一串仅包含**小写字母**的字符串 t_i。

由于锁的独特构造，妮妮可以使用任意一把钥匙打开宝箱，但是从宝箱中获得的金币数量并不相同。使用第 i 个钥匙会获得 w_i 个金币，其中 w_i 为 t_i 在 s 中出现的次数。

由于宝箱只能开一次，妮妮想知道打开一次宝箱最多可以获得多少金币。

输入格式

第一行一个字符串 s（$1 \leq |s| \leq 10^5$），表示锁上的字符串。

第二行一个正整数 m（$1 \leq m \leq 100$），表示钥匙的个数。

接下来 m 行每行一个字符串 t_i（$1 \leq |t_i| \leq 10^5$），表示第 i 个钥匙上的字符串。

输出格式

输出一行一个整数，表示妮妮打开一次宝箱最多获得的金币个数。

题目分析

对每一个待求解的子串KMP匹配。

题目求解

```
#include <bits/stdc++.h>
using namespace std;
int n, k, len1, len2;
int next1[1000001];
char s1[1000001];
char s2[1000001];
void get_next() {                    //求出next数组
    //next数组是从 S[0到i-1]前子串的前缀后缀最大值
    int t1 = 0;                      //后缀匹配指向
    int t2 = -1;                     //前缀匹配指向
    next1[0] = t2 = -1;
    while (t1 < len2) {
        // t2 == -1 就是找无可找 or 匹配成功，相同前缀长度增加1
        if (t2 == -1 || s2[t1] == s2[t2]) { //类似于KMP的匹配
            t1++;
            t2++;
            next1[t1] = t2;
        }
```

```
        // 匹配不成功则在前面的子串中继续搜索，直至找不到（即 t2 == -1 的情况）
        else {
            t2 = next1[t2];    //失配
        }
    }
}

int KMP() { //KMP
    int flag=0;
    int t1 = 0, t2 = 0;                         //从0位开始匹配
    while (t1 < len1) {                         //临界值
        //t2 == -1 找无可找，从 S[i+1] 开始和 T[0] 匹配 or 当匹配成功时，往下匹配。
        if (t2 == -1 || s1[t1] == s2[t2]) { //匹配成功，继续
            t1++;
            t2++;
        }
        //匹配不成功则用 next(t2) 找下一次匹配的位置
        else {
            t2 = next1[t2];     //失配
        }
        if (t2 == len2) {                       //t2==lenn2时，匹配成功；t1-len2+1即为第
一个字母的位置
            t2 = next1[t2];                             //不可重叠，用0
            flag++;
        }
    }           //匹配成功后，t2置为next[t2]
    return flag;
}
int main() {
    scanf("%s", s1);
    int t;
    scanf("%d", &t);
    int min=-1;
    while (t--) {
        scanf("%s", s2);
        len1 = strlen(s1);
        len2 = strlen(s2);
        get_next();
        min=max(min,KMP());
    }
    printf("%d",min);
    return 0;
}
```

时间复杂度

经分析，BF算法$O(m(len1+len2))$。