字符串 1

KMP 算法

算法简介

给定两个字符串 S_1,S_2，查询 S_2 在 S_1 中出现的位置。时间复杂度 $O(|S_1|+|S_2|)$。

算法实现

洛谷p3375

定义字符串 S 的前缀函数 f 表示 S 的最长的相等的真前缀和真后缀，先考虑计算模式串 S_2 的 f 函数。

对 $f(i+1)$，可以考虑从大到小枚举 $s[1,i]$ 的相等真前后缀，首先最大相等真前后缀为 $f(i)$，考虑下面的串

$$s_1s_2\cdots s_{f(i)}s_{f(i)+1}\cdots s_{i-f(i)+1}s_{i-f(i)+2}\cdots s_is_{i+1}$$

已知 $s[1,f(i)]=s[i-f(i)+1,i]$，于是如果 $s_{i+1}=s_{f(i)+1}$，则有 $f(i+1)=f(i)+1$。

如果 $s_{i+1}\neq s_{f(i)+1}$，则只能考虑第二长的真前后缀，暂时记为 j。

于是有 $s[1,j]=s[i-j+1,i]=s[f(i)-j+1,f(i)]$，发现 j 是 $s[1,f(i)]$ 的真前后缀，于是有 $j=f(f(i))$。

不停尝试，直到 $j=0$ 即没有满足条件真前后缀，可以停止尝试。

考虑动态维护当前真前后缀即可，j 指针每次最多增加 1，每次匹配失败至少减少 1，最多增加 $|s_2|$ 次，所以时间复杂度为 $O(|s_2|)$。

接下来考虑匹配过程，事实上匹配过程当匹配失败后直接跳到上一个相等的真前后缀继续比较即可，类似的有时间复杂度为 $O(|s_1|)$。

于是总时间复杂度 $O(|s_1|+|s_2|)$。

```
const int MAXN=1e6+5;
namespace KMP{
    int f[MAXN];
    void find(char *s1,int n,char *s2,int m){//s下标从1开始
        int pos=0;
        f[1]=0;
        _rep(i,2,m){
            while(pos&&s2[i]!=s2[pos+1])pos=f[pos];
            if(s2[i]==s2[pos+1])pos++;
            f[i]=pos;
        }
        pos=0;
```

```
        _rep(i,1,n){
            while(pos&&s1[i]!=s2[pos+1])pos=f[pos];
            if(s1[i]==s2[pos+1])pos++;
            if(pos==m){
                printf("%d\n",i-pos+1);
                pos=f[pos];
            }
        }
    }
}
char buf[MAXN],str[MAXN];
int main()
{
    scanf("%s%s",buf+1,str+1);
    int n=strlen(buf+1),m=strlen(str+1);
    KMP::find(buf,n,str,m);
    _rep(i,1,m)space(KMP::f[i]);
    return 0;
}
```

应用

循环串

考虑串 $abcabcabc$ 发现 $f(n)=6$ 去掉 $f(6)$ 长度的后缀后可以得到串 abc 发现该串恰好为 abc 为最大周期的循环串。

于是有结论若 $n-f(n)\mid n$ 则 $s[1,n]$ 为以 $s[1,n-f(n)]$ 为最大周期的循环串。

考虑串 $abcabca$ 发现 $f(7)=4$ 去掉 $f(7)$ 长度的后缀后可以得到串 abc 发现只要在串的末尾补上 bc 就可以构造以 abc 为周期的串。

于是有结论若 $n-f(n)\nmid n$ 则在 $s[1,n]$ 末尾补上长度为 $n-f(n)-(n\bmod {(n-f(n))})$ 可以构成以 $s[1,n-f(n)]$ 为最大周期的循环串。

From:
https://wiki.cvbbacm.com/ - **CVBB ACM Team**

Permanent link:
https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:legal_string:jxm2001:%E5%AD%97%E7%AC%A6%E4%B8%B2_1&rev=1598417719

Last update: **2020/08/26 12:55**