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。

根据已知，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，类似地有时间复杂度为 $O(|s_1|)$。

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。

前缀统计

考虑对串 s 统计 s 的每个前缀在 s 中的出现次数。

先考虑统计所有形如 $s[l,r](l\gt 1)$ 的串对答案的贡献，发现这些串等价于所有 $s[1,r]$ 的真后缀集合。

考虑串 $s[1,i]$ 的所有真后缀，同时他们对 $f(i),f(f(i)),f(f(f(i)))\cdots$ 的前缀产生贡献。

于是从 n 到 1 依次转移真后缀的贡献，最后补上所有前缀的贡献即可。

```
int cnt[MAXN];
```

```
void cal(int *s,int n){
    get_f(s,n);
    _rep(i,1,n)cnt[f[i]]++;
    for(int i=n;i;i--)cnt[f[i]]+=cnt[i];
    _rep(i,1,n)cnt[i]++;
}
```

接下来考虑统计串 t 的每个前缀在串 s 中的出现次数。

事实上，直接先将与每个 $s[1,i]$ 的后缀匹配的 t 的最长的前缀加入桶，然后按上述方式转移即可。

不同子串统计

一开始建立一个空串，考虑每次向空串中加入新字母。

假设当前串为 s，新加入的字母为 c，设 $t=s+c,t'=reverse(t),|t|=n$。

对串 t' 跑 KMP 算法，显然如果有 $f(i)=j$，则表示 $t'[1,j]=t'[i-j+1,i]$。

这等价于 $t[n-j+1,n]=t[n-i+1,n-i+j]$，于是 $t[n-j+1,n]$ 不是新子串。

设 $m=\max(f(i))$，于是有 $t'[1,i](i\gt m)$ 在 t' 中唯一，同样也在 t 中唯一，于是 $t[n-i+1,n](i\gt m)$ 为新子串。

算法练习

习题一

UVA11022

题意

给定一个字符串 s，求 s 压缩后的最小长度。

压缩示例：

1. $abaaba\to ab(a)^2ba$，长度 $6\to 5$，但这不是最佳方案
2. $abaaba\to (aba)^2$，长度 $6\to 3$
3. $abbabb\to (a(b)^2)^2$，长度 $6\to 2$

题解

考虑一个字符串 s，要么 $s=s_1+s_2$，要么 $s=(s_3)^k$。

考虑区间 dp，对情况一，有 $\text{dp}(l,r)=\min_{l\le i\lt r}(\text{dp}(l,i)+\text{dp}(i+1,r))$。

对情况二，考虑求出 s 的周期 d，有 $\text{dp}(l,r)=\text{dp}(l,l+d-1)$。

边界条件 $l=r,\text{dp}(l,r)=1$ 时间复杂度 $O(n^3)$

```
const int MAXN=100;
namespace KMP{
    int f[MAXN];
    int get_loop(char *s,int n){
        int pos=0;
        f[1]=0;
        _rep(i,2,n){
            while(pos&&s[i]!=s[pos+1])pos=f[pos];
            if(s[i]==s[pos+1])pos++;
            f[i]=pos;
        }
        if(!f[n]||n%(n-f[n]))return 0;
        return n-f[n];
    }
}
int dp[MAXN][MAXN];
char buf[MAXN];
int dfs(int lef,int rig){
    if(~dp[lef][rig])return dp[lef][rig];
    if(lef==rig)return 1;
    int ans=MAXN;
    _for(i,lef,rig)
    ans=min(ans,dfs(lef,i)+dfs(i+1,rig));
    int len=KMP::get_loop(buf+lef-1,rig-lef+1);
    if(len)ans=min(ans,dfs(lef,lef+len-1));
    return dp[lef][rig]=ans;
}
int main()
{
    while(~scanf("%s",buf+1)){
        if(buf[1]=='*')break;
        int n=strlen(buf+1);
        mem(dp,-1);
        enter(dfs(1,n));
    }
    return 0;
}
```

习题二

CF808G

题意

给定一个字符串 s 和模式串 t。s 只包含 26 个小写字母和 ?。t 只包含 26 个小写字母。

将 s 中的所有 ? 替换为任意小写字母，使得 t 出现次数最大。

题解

不妨设 $|s|=n,|t|=m$。

设 $\text{dp}_1(i)$ 表示 $s[1,i]$ 的所有方案中 t 出现的最大次数。$\text{dp}_2(i)$ 表示 $s[1,i]$ 的满足 s 长度为 m 的后缀为 t 的方案中 t 出现的最大次数。

首先有 $\text{dp}_1(i)=\max(\text{dp}_1(i-1),\text{dp}_2(i))$。表示 $s[1,i]$ 的长度为 m 的后缀是否为 t。

接下来考虑 $\text{dp}_2(i)$。显然如果 $s[1,i]$ 的长度为 m 的后缀无法变成 t。则 $\text{dp}_2(i)=0$。

否则考虑是否存在 $i-m\lt j\lt i$ 使得 $s[1,j]$ 的长度为 m 的后缀为 t。如果不存在 j。则显然有 $\text{dp}_2(i)=\text{dp}_1(i-m)+1$。

如果存在 j 满足上述条件，取最小的 j。则一定有 $\text{dp}_2(i)=\text{dp}_2(i-m+j)+1$。对其余所有 j。有 $\text{dp}_2(i)\ge\text{dp}_2(i-m+j)+1$。

同时对所有满足条件的 j。不难发现有 $t[1,m]$ 的长度为 j 的前后缀相同。

由于无法判断 j 是否满足上述条件，于是暴力跳 t 的 f 函数枚举每个 j 即可。总时间复杂度 $O(nm)$。

```
const int MAXN=1e5+5;
int f[MAXN];
void get_f(char *s,int n){
    int pos=0;
    f[1]=0;
    _rep(i,2,n){
        while(pos&&s[i]!=s[pos+1])pos=f[pos];
        if(s[i]==s[pos+1])pos++;
        f[i]=pos;
    }
}
char s[MAXN],t[MAXN];
int n,m,dp1[MAXN],dp2[MAXN];
bool legal(int pos){
    for(int i=pos-m+1,j=1;j<=m;i++,j++){
        if(s[i]!=t[j]&&s[i]!='?')return false;
    }
    return true;
}
int main()
{
```

```
    scanf("%s%s",s+1,t+1);
    n=strlen(s+1),m=strlen(t+1);
    get_f(t,m);
    _rep(i,1,n){
        dp1[i]=dp1[i-1];
        if(legal(i)){
            dp2[i]=dp1[i-m]+1;
            for(int j=f[m];j;j=f[j])
            dp2[i]=max(dp2[i],dp2[i-m+j]+1);
            dp1[i]=max(dp1[i],dp2[i]);
        }
    }
    enter(dp1[n]);
    return 0;
}
```

Z 函数

算法简介

给定两个字符串 S_1,S_2，查询 S_1,S_2 与 S_2 每个后缀的 LCP，时间复杂度 $O(|S_1|+|S_2|)$。

算法实现

设 $z(i)$ 表示 $S[1,|S|],S[i,|S|]$ 的 LCA 长度，先考虑求模式串 S_2 的 z 函数。

假设 $i\lt l$ 的 $z(i)$ 已知，考虑求 $z(l)$，直接暴力拓展 $[l,r]$，直到取到 $s_r=s_{r-l+1}$ 的最大值，易知此时 $z(l)=r-l+1$。

接下来考虑区间 $i\in [l+1,r]$，首先由于 $s_{r+1}\neq s_{r-l+2}$，于是 $z(i)\le r-i+1$。

同时由于 $s[l,r]$ 与 $s[1,r-l+1]$ 完全相同，于是若 $z(i-l+1)\le r-i+1$，显然有 $z(i)=z(i-l+1)$，否则有 $z(i)=r-i+1$。

于是可以在 $O(r-l)$ 时间求出 $r-l+1$ 个 z 函数的值，于是求 z 函数的时间复杂度为 $O(|S_2|)$。

接下来考虑求 S_1 与 S_2 每个后缀的 LCP，同样可以按上述方法求取，时间复杂度 $O(|S_1|)$，于是总时间复杂度 $O(|S_1|+|S_2|)$。

```
const int MAXN=2e7+5;
namespace KMP{
    int z[MAXN],p[MAXN];
    void cmp(char *s1,int n,char *s2,int m){//s下标从1开始
        z[1]=m;
        _rep(i,2,m)z[i]=0;
```

```
        for(int i=2,l=0,r=0;i<=m;i++){
            if(i<=r)z[i]=min(z[i-l+1],r-i+1);
            while(i+z[i]<=m&&s2[z[i]+1]==s2[i+z[i]])z[i]++;
            if(i+z[i]-1>r)l=i,r=i+z[i]-1;
        }
        _rep(i,1,n)p[i]=0;
        for(int i=1,l=0,r=0;i<=n;i++){
            if(i<=r)p[i]=min(z[i-l+1],r-i+1);
            while(i+p[i]<=n&&s2[p[i]+1]==s1[i+p[i]])p[i]++;
            if(i+p[i]-1>r)l=i,r=i+p[i]-1;
        }
    }
}
char a[MAXN],b[MAXN];
int main()
{
    gets(a+1);gets(b+1);
    int n=strlen(a+1),m=strlen(b+1);
    KMP::cmp(a,n,b,m);
    LL ans1=0,ans2=0;
    _rep(i,1,m)
    ans1^=1LL*i*(KMP::z[i]+1);
    _rep(i,1,n)
    ans2^=1LL*i*(KMP::p[i]+1);
    enter(ans1);enter(ans2);
    return 0;
}
```

算法练习

习题一

CF126B

题意

给定字符串 S 求既是 S 的前缀又是 S 的后缀同时又在 S 中间出现过的最长子串。

题解 1

考虑求出每个位置的 z 函数，然后找到与前缀相同的在中间出现过的最长字串，有 $\text{pre}=\max_{1\lt i \lt n}(\min(z(i),n-i))$

接下来从大到小枚举每个与前缀相同的后缀，判定条件为 $i+z(i)-1=n$ 如果该后缀长度小于 pre 则得到答案。时间复杂度 $O(|S|)$

```
const int MAXN=1e6+5;
namespace KMP{
    int z[MAXN];
    void get_z(char *s,int n){
        z[1]=n;
        _rep(i,2,n)z[i]=0;
        for(int i=2,l=0,r=0;i<=n;i++){
            if(i<=r)z[i]=min(z[i-l+1],r-i+1);
            while(i+z[i]<=n&&s[z[i]+1]==s[i+z[i]])z[i]++;
            if(i+z[i]-1>r)l=i,r=i+z[i]-1;
        }
    }
}
char s[MAXN];
int main()
{
    gets(s+1);
    int n=strlen(s+1);
    KMP::get_z(s,n);
    int pre=0,ans=0;
    _for(i,2,n)pre=max(pre,min(KMP::z[i],n-i));
    _rep(i,2,n)if(i+KMP::z[i]-1==n&&KMP::z[i]<=pre){
        ans=KMP::z[i];
        break;
    }
    if(!ans)puts("Just a legend");
    else
    _rep(i,1,ans)putchar(s[i]);
    return 0;
}
```

题解 2

考虑求出每个位置的 f 函数，同样找到与前缀相同的在中间出现过的最长字串，有 $\text{pre}=\max_{1\lt i \lt n}f(i)$。

接下来从大到小枚举每个与前缀相同的后缀，初始为 $f(n)$。然后不断迭代 f 即可。如果该后缀长度小于 pre 则得到答案。时间复杂度 $O(|S|)$。

习题二

CF432D

题意

给你一个长度为 n 的字符串。定义 “ 完美子串 ” 既是它的前缀也是它的后缀，求 “ 完美子串 ” 的个数且

统计这些子串的在字符串中出现的次数。

题解 1

考虑求出每个位置的 z 函数，然后找到与前缀相同的在中间出现过的最长字串，有 $\text{pre}=\max_{1\lt i \lt n}(\min(z(i),n-i))$。

接下来从大到小枚举每个与前缀相同的后缀，判定条件为 $i+z(i)-1=n$。如果该后缀长度小于 pre 则得到答案。时间复杂度 $O(|S|)$。

题解 2

考虑求出每个位置的 f 函数，统计前缀出现次数。

然后从大到小枚举每个与前缀相同的后缀并记录答案，初始为 $f(n)$。然后不断迭代 f 即可。时间复杂度 $O(|S|)$。

```
const int MAXN=1e6+5;
namespace KMP{
    int f[MAXN];
    void get_f(char *s,int n){
        int pos=0;
        f[1]=0;
        _rep(i,2,n){
            while(pos&&s[i]!=s[pos+1])pos=f[pos];
            if(s[i]==s[pos+1])pos++;
            f[i]=pos;
        }
    }
}
char s[MAXN];
LL c[MAXN];
int main()
{
    gets(s+1);
    int n=strlen(s+1);
    KMP::get_f(s,n);
    _rep(i,1,n)c[KMP::f[i]]++;
    LL ans=0;
    stack<pair<int,LL> >pt;
    for(int i=n;i;i--)c[KMP::f[i]]+=c[i];
    for(int i=n;i;i=KMP::f[i])
    pt.push(make_pair(i,c[i]+1));
    enter(pt.size());
    while(!pt.empty()){
        printf("%d %lld\n",pt.top().first,pt.top().second);
        pt.pop();
    }
```

```
    return 0;
}
```

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=1598498526

Last update: **2020/08/27 11:22**