# KMP算法

#include <cstdio>

#include <cstring>

#include <algorithm>

using namespace std;

#define N 1000010

int nxt[N];

char S[N],T[N];

void get\_next(char T[N],int tlen)

{

int i=0,j=-1;

nxt[0]=-1;

while(i<tlen)

if(j==-1||T[i]==T[j]) nxt[++i]=++j;

else j=nxt[j];

}

/\*

返回模式串T在主串S中首次出现的位置

返回的位置是从0开始的。

\*/

int kmp\_index(char S[],int slen,char T[],int tlen)

{

get\_next(T,tlen);

int j=0;

for(int i=0;i<slen;++i)

{

while(j>0&&S[i]!=T[j]) j=nxt[j];

if(S[i]==T[j]) ++j;

if(j==tlen) return i-tlen+1;

}

return -1;

}

/\*

返回模式串T在主串S中出现的次数

\*/

int kmp\_count(char S[],int slen,char T[],int tlen)

{

get\_next(T,tlen);

int ans=0,j=0;

for(int i=0;i<slen;++i)

{

while(j>0&&S[i]!=T[j]) j=nxt[j];

if(S[i]==T[j]) ++j;

if(j==tlen)

{

++ans;

j=nxt[j];//子串可重叠

//j=0;//子串不可重叠

}

}

return ans;

}

# 扩展KMP算法

/\*\*

拓展kmp是对KMP算法的扩展，它解决如下问题：

定义母串S，和字串T，设S的长度为n，T的长度为m，

求T与S的每一个后缀的最长公共前缀，

也就是说，设extend数组,extend[i]表示T与S[i,n-1]的最长公共前缀，

要求出所有extend[i](0<=i<n)，

设辅助数组nxt[i]表示T[i,m-1]和T的最长公共前缀长度

\*\*/

#include<iostream>

#include<string>

#include<cstring>

#include<cstdio>

using namespace std;

const int N=100010;

int nxt[N],extand[N];

char S[N],T[N];

void get\_next(char T[],int tlen)

{

int a=0;

nxt[0]=tlen;

while(a<tlen-1 && T[a]==T[a+1]) a++;

nxt[1]=a;

a=1;

for(int k=2; k<tlen; k++)

{

int p=a+nxt[a]-1,L=nxt[k-a];

if( (k-1)+L >= p)

{

int j = (p-k+1)>0 ? (p-k+1) : 0;

while(k+j<tlen && T[k+j]==T[j]) j++;

nxt[k]=j;

a=k;

}

else

nxt[k]=L;

}

}

void get\_extand(char S[],int slen,char T[],int tlen)

{

get\_next(T,tlen);

int a=0;

int min\_len = slen < tlen ? slen : tlen;

while(a<min\_len && S[a]==T[a]) a++;

extand[0]=a;

a=0;

for(int k=1; k<slen; k++)

{

int p=a+extand[a]-1, L=nxt[k-a];

if( (k-1)+L >= p)

{

int j= (p-k+1) > 0 ? (p-k+1) : 0;

while(k+j<slen && j<tlen && S[k+j]==T[j]) j++;

extand[k]=j;

a=k;

}

else

extand[k]=L;

}

}

int main(void)

{

while(scanf("%s%s",S,T)==2)

{

int slen=strlen(S),tlen=strlen(T);

get\_extand(S,slen,T,tlen);

for(int i=0; i<tlen; i++)

printf("%d ",nxt[i]);

puts("");

for(int i=0; i<slen; i++)

printf("%d ",extand[i]);

puts("");

}

return 0;

}

# manacer算法

/\*

manacher：可以解决最长回文问题。

算法：

首先，将字符串的每个字符左右加入#，

并在s0位置加入\*（如果字符串中本身含有这些，则换成未出现过的字符），

此时字符串的长度为len+len+3，

即加入了len+1个#和一个\*; （比如：aba变成 \*#a#b#a#）

得到一个p数组，该数组是基于新字符串进行的，

p[i]表示以 i 为中心的最长回文半径

p[i] - 1正好是原字符串中最长回文串的长度，如

S \* # a # b # b # a # h # o # p # x # p #

p 1 2 1 2 5 2 1 2 1 2 1 2 1 2 1 4 1 2 1

\*/

void manacher(char S[],int len) //manacher 函数

{

for(int i=len;i>=0;--i) //将s扩大，中间加#，开头加\*

{

S[i+i+2]=S[i];

S[i+i+1]='#';

}

S[0]='\*';

int id,mx=0; //mx代表以id为中心时，到达最远的位置

for(int i=1;i<len+len+1;++i)

{

if(mx>i) p[i]=min(p[2\*id-i],mx-i); //如果到达最远位置大于当前匹配的地方，则p[i]取min（id的对称点的p，到达最远距离-i）

else p[i]=1; //如果i在mx右方，则p[i]=-1;

while(S[i-p[i]] == S[i+p[i]])++p[i]; //判断i回文长度

if(i+p[i]>mx) //看是否要更新最远距离，如果要，将此点作为中心。

{

id=i;

mx=p[i]+i;

}

}

}

# 最小表示法和最大表示法

/\*

循环字符串的最小表示法的问题可以这样描述：

对于一个字符串S，求S的循环的同构字符串S’中字典序最小的一个。

\*/

//返回字符串S'的首字母的id

int get\_min(char s[],int len)

{

int i = 0, j = 1, l;

while(i < len && j < len)

{

for(l = 0; l < len; l++)

if(s[(i + l) % len] != s[(j + l) % len]) break;

if(l >= len) break;

if(s[(i + l) % len] > s[(j + l) % len])

{

if(i + l + 1 > j) i = i + l + 1;

else i = j + 1;

}

else if(j + l + 1 > i) j = j + l + 1;

else j = i + 1;

}

return i < j ? i : j;

}

//返回字符串S'的首字母的id

int get\_max(char s[],int len)

{

int i = 0, j = 1, k = 0;

while(i < len && j < len && k < len)

{

int t = s[(i+k)%len]-s[(j+k)%len];

if(!t) k++;

else

{

if(t > 0)

{

if(j+k+1 > i) j = j+k+1;

else j = i+1;

}

else if(i+k+1 > j) i = i+k+1;

else i = j+1;

k = 0;

}

}

return i < j ? i : j;

}