后缀数组资料

例 8 ：重复次数最多的连续重复子串 (spoj687,pku3693)

例 10: 长度不小于 k的公共子串的个数 (pku3415)

例 11: 不小于k个字符串中的最长子串 (pku3294)

[例6：最长回文子串（ural1297）](http://blog.csdn.net/jarily/article/details/8711009)

[例7：连续重复子串(pku2406)](http://blog.csdn.net/hnust_xiehonghao/article/details/9150273)

例8：[重复次数最多的连续重复子串](http://m.blog.csdn.net/blog/yanglei040/21179865)(spoj687,pku3693)

例9：最长公共子串(pku2774,ural1517)

**[PKU2774(Long Long Message)求最长公共子串(后缀数组)](http://blog.csdn.net/jarily/article/details/8709964)**

[POI 2000 Repetitions 最长公共子串](http://blog.csdn.net/self_chou/article/details/7761180)

例10:长度不小于k 的公共子串的个数(pku3415)

http://m.blog.csdn.net/blog/kalilili/44786949

例11:不小于k 个字符串中的最长子串([pku3294](http://blog.sina.com.cn/s/blog_6635898a0102e0oi.html))



详细资料：

[例6：最长回文子串（ural1297）](http://blog.csdn.net/jarily/article/details/8711009)

**[Ural1297(Palindrome)求最长回文子串(后缀数组)](http://blog.csdn.net/jarily/article/details/8711009)**

题目地址：

http://acm.timus.ru/problem.aspx?space=1&num=1297

题目大意：

给定一个字符串,求最长回文子串;

算法分析：

穷举每一位，然后计算以这个字符为中心的最长回文子串;

注意这里要分两种情况，一是回文子串的长度为奇数，二是长度为偶数;

两种情况都可以转化为求一个后缀和一个反过来写的后缀的最长公共前缀;

即将整个字符串反过来写在原字符串后面，中间用一个特殊的字符隔开;

这样就把问题变为了求这个新的字符串的某两个后缀的最长公共前缀;

根据利用后缀数组sa求出的height数组的性质;

可以求两个后缀的最长公共前缀可以转化为求某个区间上的最小值;

即求区间最值问题(RMQ问题);

问题补充：

后缀是指从某个位置i开始到整个串末尾结束的一个特殊子串;

字符串r的从第i个字符开始的后缀表示为Suffix(i),也就是Suffix(i)=r[i...len(r)];

后缀数组保存的是一个字符串的所有后缀的排序结果;

其中SA[i]保存的是字符串所有的后缀中第i小的后缀的开头位置;

定义height[i]=suffix(sa[i-1])和suffix(sa[i])的最长公共前缀，

也就是排名相邻的两个后缀的最长公共前缀;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<iostream>

#include<cstring>

#include<cstdlib>

#include<cstdio>

#include<climits>

#include<algorithm>

**using** **namespace** std;

**const** **int** N=2010;

**const** **int** M=128;//r数组的最大值

#define F(x) ((x)/3+((x)%3==1?0:tb))

#define G(x) ((x)<tb?(x)\*3+1:((x)-tb)\*3+2)

**int** wa[N],wb[N],wv[N],\_ws[N];

**int** c0(**int** \*r,**int** a,**int** b){

**return** r[a]==r[b]&&r[a+1]==r[b+1]&&r[a+2]==r[b+2];

}

**int** c12(**int** k,**int** \*r,**int** a,**int** b){

**if**(k==2)

**return** r[a]<r[b]||r[a]==r[b]&&c12(1,r,a+1,b+1);

**else**

**return** r[a]<r[b]||r[a]==r[b]&&wv[a+1]<wv[b+1];

}

**void** sort(**int** \*r,**int** \*a,**int** \*b,**int** n,**int** m){

**for**(**int** i=0; i<n; i++)

wv[i]=r[a[i]];

**for**(**int** i=0; i<m; i++)

\_ws[i]=0;

**for**(**int** i=0; i<n; i++)

\_ws[wv[i]]++;

**for**(**int** i=1; i<m; i++)

\_ws[i]+=\_ws[i-1];

**for**(**int** i=n-1; i>=0; i--)

b[--\_ws[wv[i]]]=a[i];

**return**;

}

**void** dc3(**int** \*r,**int** \*sa,**int** n,**int** m){

**int** i,j,\*rn=r+n,\*san=sa+n,ta=0,tb=(n+1)/3,tbc=0,p;

r[n]=r[n+1]=0;

**for**(i=0; i<n; i++){

**if**(i%3!=0)

wa[tbc++]=i;

}

sort(r+2,wa,wb,tbc,m);

sort(r+1,wb,wa,tbc,m);

sort(r,wa,wb,tbc,m);

**for**(p=1,rn[F(wb[0])]=0,i=1; i<tbc; i++){

rn[F(wb[i])]=c0(r,wb[i-1],wb[i])?p-1:p++;

}

**if**(p<tbc)

dc3(rn,san,tbc,p);

**else** {

**for**(i=0; i<tbc; i++)

san[rn[i]]=i;

}

**for**(i=0; i<tbc; i++){

**if**(san[i]<tb)

wb[ta++]=san[i]\*3;

}

**if**(n%3==1)

wb[ta++]=n-1;

sort(r,wb,wa,ta,m);

**for**(i=0; i<tbc; i++)

wv[wb[i]=G(san[i])]=i;

**for**(i=0,j=0,p=0; i<ta && j<tbc; p++){

sa[p]=c12(wb[j]%3,r,wa[i],wb[j])?wa[i++]:wb[j++];

}

**for**(; i<ta; p++)

sa[p]=wa[i++];

**for**(; j<tbc; p++)

sa[p]=wb[j++];

**return**;

}

**int** rank[N],height[N];

**void** calheight(**int** \*r,**int** \*sa,**int** n){

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

**for**(i=1; i<=n; i++)

rank[sa[i]]=i;

**for**(i=0; i<n; height[rank[i++]]=k){

**for**(k?k--:0,j=sa[rank[i]-1]; r[i+k]==r[j+k]; k++);

}

**return**;

}

**int** RMQ[N];

**int** mm[N];

**int** best[20][N];

**void** initRMQ(**int** n){

mm[0]=-1;

**for**(**int** i=1; i<=n; i++){

mm[i]=((i&(i-1))==0)?mm[i-1]+1:mm[i-1];

}

**for**(**int** i=1; i<=n; i++)

best[0][i]=i;

**for**(**int** i=1; i<=mm[n]; i++)

**for**(**int** j=1; j<=n+1-(1<<i); j++){

**int** a=best[i-1][j];

**int** b=best[i-1][j+(1<<(i-1))];

**if**(RMQ[a]<RMQ[b])

best[i][j]=a;

**else**

best[i][j]=b;

}

**return**;

}

**int** askRMQ(**int** a,**int** b){

**int** t=mm[b-a+1];

b-=(1<<t)-1;

a=best[t][a];

b=best[t][b];

**return** RMQ[a]<RMQ[b]?a:b;

}

**int** lcp(**int** a,**int** b){

a=rank[a];

b=rank[b];

**if**(a>b)

{

**int** t=a;

a=b;

b=t;

}

**return**(height[askRMQ(a+1,b)]);

}

**char** st[N];

**int** r[N\*3],sa[N\*3];

**int** main(){

scanf("%s",st);

**int** len=strlen(st);

**for**(**int** i=0; i<len; i++)

r[i]=st[i];

r[len]=1;

**for**(**int** i=0; i<len; i++)

r[i+len+1]=st[len-1-i];

**int** n=len+len+1;

r[n]=0;

dc3(r,sa,n+1,M);

calheight(r,sa,n);

**for**(**int** i=1; i<=n; i++)

RMQ[i]=height[i];

initRMQ(n);

**int** ans=0;

**int** w;

**for**(**int** i=0; i<len; i++){

**int** k=lcp(i,n-i);

**if**(k\*2>ans){

ans=k\*2;

w=i-k;

}

k=lcp(i,n-i-1);

**if**(k\*2-1>ans){

ans=k\*2-1;

w=i-k+1;

}

}

st[w+ans]=0;

printf("%s\n",st+w);

**return** 0;

}

[例7：连续重复子串(pku2406)](http://blog.csdn.net/hnust_xiehonghao/article/details/9150273)

[poj 2406 求连续重复子串出现的次数 后缀数组](http://blog.csdn.net/hnust_xiehonghao/article/details/9150273)

|  |  |  |
| --- | --- | --- |
| **Time Limit:** 3000MS |  | **Memory Limit:** 65536K |
| **Total Submissions:** 26591 |  | **Accepted:** 11130 |

Description

Given two strings a and b we define a\*b to be their concatenation. For example, if a = "abc" and b = "def" then a\*b = "abcdef". If we think of concatenation as multiplication, exponentiation by a non-negative integer is defined in the normal way: a^0 = "" (the empty string) and a^(n+1) = a\*(a^n).

Input

Each test case is a line of input representing s, a string of printable characters. The length of s will be at least 1 and will not exceed 1 million characters. A line containing a period follows the last test case.

Output

For each s you should print the largest n such that s = a^n for some string a.

Sample Input

abcd

aaaa

ababab

.

Sample Output

1

4

3

Hint

This problem has huge input, use scanf instead of cin to avoid time limit exceed.

Source

[Waterloo local 2002.07.01](http://poj.org/searchproblem?field=source&key=Waterloo+local+2002.07.01)

[[Submit](http://poj.org/submit?problem_id=2406)] [[Go Back](http://blog.csdn.net/hnust_xiehonghao/article/details/9150273)] [[Status](http://poj.org/problemstatus?problem_id=2406)] [[Disc](http://poj.org/bbs?problem_id=2406)

<http://poj.org/problem?id=2406>

问题描述：给定一个字符串L，已知这个字符串是由某个字符串S重复R次而得到的， 求R的最大值。

方法一：后缀数组。

从长度为1开始枚举到长度为n，如果n%i==0，那么判断LCS (suff(i+0),suff(0))是否等于n-i。

根据h可以求得LCS，其中lcs(i,j)=min{h[rank[i]+1],...,h[rank[j]]}，其中假设rank[i]<rank[j]。

#include <stdio.h>

#include<string.h>

#define maxn 1000001

**char** c;

**int** r[maxn\*3],sa[maxn\*3];

**int** ans[maxn];

**char** str[maxn\*3];

#define F(x) ((x)/3+((x)%3==1?0:tb))

#define G(x) ((x)<tb?(x)\*3+1:((x)-tb)\*3+2)

**int** wa[maxn],wb[maxn],wv[maxn],ws[maxn];

**int** c0(**int** \*r,**int** a,**int** b)

{**return** r[a]==r[b]&&r[a+1]==r[b+1]&&r[a+2]==r[b+2];}

**int** c12(**int** k,**int** \*r,**int** a,**int** b)

{**if**(k==2) **return** r[a]<r[b]||r[a]==r[b]&&c12(1,r,a+1,b+1);

**else** **return** r[a]<r[b]||r[a]==r[b]&&wv[a+1]<wv[b+1];}

**void** sort(**int** \*r,**int** \*a,**int** \*b,**int** n,**int** m)

{

**int** i;

**for**(i=0;i<n;i++) wv[i]=r[a[i]];

**for**(i=0;i<m;i++) ws[i]=0;

**for**(i=0;i<n;i++) ws[wv[i]]++;

**for**(i=1;i<m;i++) ws[i]+=ws[i-1];

**for**(i=n-1;i>=0;i--) b[--ws[wv[i]]]=a[i];

**return**;

}

**void** dc3(**int** \*r,**int** \*sa,**int** n,**int** m) // r为待匹配数组 n为总长度 m为字符范围

{

**int** i,j,\*rn=r+n,\*san=sa+n,ta=0,tb=(n+1)/3,tbc=0,p;

r[n]=r[n+1]=0;

**for**(i=0;i<n;i++) **if**(i%3!=0) wa[tbc++]=i;

sort(r+2,wa,wb,tbc,m);

sort(r+1,wb,wa,tbc,m);

sort(r,wa,wb,tbc,m);

**for**(p=1,rn[F(wb[0])]=0,i=1;i<tbc;i++)

rn[F(wb[i])]=c0(r,wb[i-1],wb[i])?p-1:p++;

**if**(p<tbc) dc3(rn,san,tbc,p);

**else** **for**(i=0;i<tbc;i++) san[rn[i]]=i;

**for**(i=0;i<tbc;i++) **if**(san[i]<tb) wb[ta++]=san[i]\*3;

**if**(n%3==1) wb[ta++]=n-1;

sort(r,wb,wa,ta,m);

**for**(i=0;i<tbc;i++) wv[wb[i]=G(san[i])]=i;

**for**(i=0,j=0,p=0;i<ta && j<tbc;p++)

sa[p]=c12(wb[j]%3,r,wa[i],wb[j])?wa[i++]:wb[j++];

**for**(;i<ta;p++) sa[p]=wa[i++];

**for**(;j<tbc;p++) sa[p]=wb[j++];

**return**;

}

**int** rank[maxn],height[maxn];

**void** calheight(**int** \*r,**int** \*sa,**int** n) // 求height数组。

{

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

**for**(i=1;i<=n;i++) rank[sa[i]]=i;

**for**(i=0;i<n;height[rank[i++]]=k)

**for**(k?k--:0,j=sa[rank[i]-1];r[i+k]==r[j+k];k++);

**return**;

}

**int** RMQ[maxn];

**int** mm[maxn];

///int best[20][maxn];//best[i][j] 表示从j开始的长度为2的i次方的一段元素的最小值

/\*void initRMQ(int n)///O(Nlogn) 预处理

{

int i,j,a,b;

for(mm[0]=-1,i=1;i<=n;i++)

mm[i]=((i&(i-1))==0)?mm[i-1]+1:mm[i-1];

for(i=1;i<=n;i++) best[0][i]=i;

for(i=1;i<=mm[n];i++)

for(j=1;j<=n+1-(1<<i);j++)

{

a=best[i-1][j];

b=best[i-1][j+(1<<(i-1))];

if(RMQ[a]<RMQ[b]) best[i][j]=a;

else best[i][j]=b;

}

return;

}

int askRMQ(int a,int b)///询问a,b后缀的最长公共前缀 O(1)查询

{

int t;

t=mm[b-a+1];b-=(1<<t)-1;

a=best[t][a];b=best[t][b];

return RMQ[a]<RMQ[b]?a:b;

}

int lcp(int a,int b)

{

int t;

a=rank[a];b=rank[b];

if(a>b) {t=a;a=b;b=t;}

return(height[askRMQ(a+1,b)]);

}

\*/

**int** f[maxn];//f[i]表示lcp(0,i);

**void** get\_f(**int** n)

{

**int** i,j,mmin;

j=rank[0];

mmin=999999999;

/\*以下2个循环内的代码顺序不同的原因是 i和j的最长公共前缀lcp(rank[i],rank[j])的值应为

rmq(height,rank[i]+1,rank[j]) 注意有个+1

\*/

**for**(i=j;i>=1;i--)

{

f[i]=mmin;

mmin=mmin<height[i]?mmin:height[i];//应该包括height[j]

}

mmin=999999999;

**for**(i=j+1;i<=n;i++)

{

mmin=mmin<height[i]?mmin:height[i]; //不应该包括height[j]

f[i]=mmin;

}

}

**int** main()

{

**int** i,n;

**while**(scanf("%s",str)!=EOF)

{

n=strlen(str);

**if**(n==1&&str[0]=='.') **break**;

**for**(i=0;i<n;i++) r[i]=str[i]-'a'+1;

r[n]=0;

dc3(r,sa,n+1,123);//千万注意+1

calheight(r,sa,n);

// initRMQ(n);

/\*

for(i=0; i<n+1; i++) // rank[i] : suffix(i)排第几

printf("rank[%d] = %d\n",i,rank[i]);

printf("\n");

for(i=0; i<n+1; i++) // sa[i] : 排在第i个的是谁

printf("sa[%d] = %d\n",i,sa[i]);

\*/

**int** len;

**int** mmax=0;

get\_f(n);

**for**(len=1;len<=n;len++)

{

**if**(n%len==0)

{

**if**(f[rank[len]]==(n-len))

///注意是rank[len],因为这里在求0和0+len的lcp ，即要求rank[0]到rank[len]之间的最小height值

{

mmax=n/len;

**break**;

}

}

}

**if**(mmax!=0)

printf("%d\n",mmax);

**else** printf("1\n");

}

**return** 0;

}

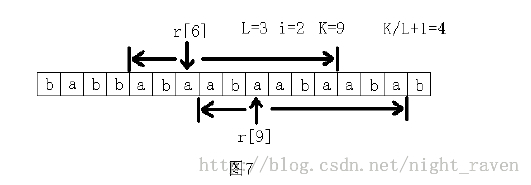
例8：[重复次数最多的连续重复子串](http://m.blog.csdn.net/blog/yanglei040/21179865)(spoj687,pku3693)

**[[原]poj3693 ,spoj687 重复次数最多的连续重复子串 后缀数组](http://m.blog.csdn.net/blog/yanglei040/21179865)**

给定一个字符串，求重复次数最多的连续重复子串。

09年罗穗骞的论文里有讲这题，先引用一下论文里的讲解：

先穷举长度L，然后求长度为L 的子串最多能连续出现几次。首先连续出现1 次是肯定可以的，所以这里只考虑至少2 次的情况。假设在原字符串中连续出现2 次，记这个子字符串为S，那么S 肯定包括了字符r[0], r[L], r[L\*2],r[L\*3], ……中的某相邻的两个。所以只须看字符r[L\*i]和r[L\*(i+1)]往前和往后各能匹配到多远，记这个总长度为K，那么这里连续出现了K/L+1 次。最后看最大值是多少。如图7 所示。



穷举长度L 的时间是n，每次计算的时间是n/L。所以整个做法的时间复杂度是O(n/1+n/2+n/3+……+n/n)=O(nlogn)。

以上内容提供了一个思路方向，但剩下的东西也挺难想的...论文里只讲了右面匹配的情况，队向左匹配的方法并没有细说（难道是认为太简单了么....= =.....），举个例子吧（数据是从poj discuss里搬来的）

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

b a c c d b a c c d b a c b d b a c b d

加入我现在枚举到了l=5,p1=5,p2=10

令k=LCS（p1,p2）=3(bac),

而重复的次数即c=k/l+1=1;

由于本题要输出字典序最小的答案，所以p1-l+1 -----p1-1之间任何一个位置都可能是我们要的答案，所以我们必须在字符匹配的情况下逐位向前平移去找答案，

移动两位后，p1=3,p2=8,此时k也增加2等于5，发现达到了l，所以此时c++,等于2，之后发现再往前平移字符依然是匹配的，所以继续，找到p1=1,p2=6,此时k=7，c=2，重复的次数和p1在3的时候没有变化，但显然此时的字典序是更小的，所以在这里更新答案的位置，下一位字符不再匹配，所以到此结束。

这样按照论文中的做法，把长度枚举一边即可得到答案。要说一下的是，这么去找最小的字典序的话，最差的情况其实还是n^2的（比如构造50000个a，跑出来结果就要将近10s），可能是数据给的太弱了（网上搜了几个题解，好多代码都被discuss里给的数据cha掉了...），这么写再poj上只用了400ms。我也没什么别的好方法了...还望大神赐教...

spoj687这题就好办了，只要求输出最大的重复次数，和上面的方法一样。枚举长度，然后枚举i\*l,(i+1)\*l的位置，由于不求字符串，这里我们便可以省去平移。求出k之后，若k%l>0的话，说明p1,p2之前的位置也可能有匹配的情况，所以找到p3=p1-(l-k%l),若此时LCS(p3,p3+l)>k，那么说明前面这l-k%l位也是匹配的，所以k+=(l-k%l)。此时c=k/l+1就是我们要的答案。

贴两个代码了，

poj3693

#include <iostream>

#include <cstdio>

#include <algorithm>

#include <string>

#include <cstring>

using namespace std;

typedef long long ll;

const int maxn=220100;

int s[maxn],ss[maxn];

int sa[maxn],t[maxn],t2[maxn],c[maxn];

int sa2[maxn];

char s1[maxn],s2[maxn];

int rank[maxn],height[maxn];

int rank2[maxn],height2[maxn];

int l1,l2;

int len,len1;

int n,m;

inline int idx(char s)

{

return s-'a'+1;

}

void getheight(int n)

{

int i,j,k=0;

for (i=0; i<=n; i++) rank[sa[i]]=i;

for (i=0; i<n; i++)

{

if (k) k--;

int j=sa[rank[i]-1];

while(s[i+k]==s[j+k]) k++;

height[rank[i]]=k;

}

}

void build\_ss(int m,int n)

{

n++;

int i,\*x=t,\*y=t2;

for (int i=0; i<m; i++) c[i]=0;

for (int i=0; i<n; i++) c[x[i]=s[i]]++;

for (int i=1; i<m; i++) c[i]+=c[i-1];

for (int i=n-1; i>=0; i--)

sa[--c[x[i]]]=i;

for (int k=1; k<=n; k<<=1)

{

int p=0;

for (i=n-k; i<n; i++) y[p++]=i;

for (i=0; i<n; i++) if (sa[i]>=k) y[p++]=sa[i]-k;

for (i=0; i<m; i++) c[i]=0;

for (i=0; i<n; i++) c[x[y[i]]]++;

for (i=1; i<m; i++) c[i]+=c[i-1];

for (i=n-1; i>=0; i--) sa[--c[x[y[i]]]] = y[i];

swap(x,y);

p=1;

x[sa[0]]=0;

for (i=1; i<n; i++)

x[sa[i]]=(y[sa[i-1]]==y[sa[i]] && y[sa[i-1]+k]==y[sa[i]+k])? p-1 : p++;

if (p>=n) break;

m=p;

}

}

int d[maxn][22];

void RMQ\_init()

{

for (int i=1; i<=n; i++) d[i][0]=height[i];

for (int j=1; (1<<j)<=n; j++)

for (int i=1; (i+(1<<j)-1)<=n; i++)

d[i][j]=min(d[i][j-1],d[i+(1<<(j-1))][j-1]);

}

int RMQ(int L,int R)

{

if (L>R) swap(L,R);

L++;

int k=0;

while((1<<(k+1))<=R-L+1) k++;

return min(d[L][k],d[R-(1<<k)+1][k]);

}

int main()

{

// freopen("in.txt","r",stdin);

int tt=0;

while(~scanf("%s",s1) && s1[0]!='#')

{

n=strlen(s1);

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

s[i]=idx(s1[i]);

s[n]=0;

build\_ss(30,n);

getheight(n);

RMQ\_init();

int ans=0,pos=-1,len=0;

for (int l=1; l<=n/2; l++){

for (int i=0; (i+1)\*l<n; i++){

int p1=i\*l,p2=(i+1)\*l;

int k=RMQ(rank[p1],rank[p2]);

int c=k/l+1;

int r=l-k%l;

int p=i\*l;

int cnt=0;

for (int j=p1-1; j>p1-l&&s[j]==s[j+l]&&j>=0; j--){

cnt++;

if (cnt==r){

c++;

p=j;

}

if (rank[j]<rank[p])

p=j;

}

if (ans<c){

pos=p;

ans=c;

len=l\*c;

}

else if (ans==c){

if (rank[pos]>rank[p]){

pos=p;

len=l\*c;

}

}

}

}

cout<<"Case "<<++tt<<": ";

for (int i=0,j=pos; i<len; i++,j++)

cout<<s1[j];

cout<<endl;

}

}

spoj687的话，上面的代码也能过，只不过要跑10s+，若用删掉平移的过程，可以把时间降到4s+

#include <iostream>

#include <cstdio>

#include <algorithm>

#include <string>

#include <cstring>

using namespace std;

typedef long long ll;

const int maxn=120100;

int s[maxn],ss[maxn];

int sa[maxn],t[maxn],t2[maxn],c[maxn];

char s1[maxn],s2[maxn];

int rank[maxn],height[maxn];

int l1,l2;

int len,len1;

int n,m;

inline int idx(char s)

{

return s-'a'+1;

}

void getheight(int n)

{

int i,j,k=0;

for (i=0; i<=n; i++) rank[sa[i]]=i;

for (i=0; i<n; i++)

{

if (k) k--;

int j=sa[rank[i]-1];

while(s[i+k]==s[j+k]) k++;

height[rank[i]]=k;

}

}

void build\_ss(int m,int n)

{

n++;

int i,\*x=t,\*y=t2;

for (int i=0; i<m; i++) c[i]=0;

for (int i=0; i<n; i++) c[x[i]=s[i]]++;

for (int i=1; i<m; i++) c[i]+=c[i-1];

for (int i=n-1; i>=0; i--)

sa[--c[x[i]]]=i;

for (int k=1; k<=n; k<<=1)

{

int p=0;

for (i=n-k; i<n; i++) y[p++]=i;

for (i=0; i<n; i++) if (sa[i]>=k) y[p++]=sa[i]-k;

for (i=0; i<m; i++) c[i]=0;

for (i=0; i<n; i++) c[x[y[i]]]++;

for (i=1; i<m; i++) c[i]+=c[i-1];

for (i=n-1; i>=0; i--) sa[--c[x[y[i]]]] = y[i];

swap(x,y);

p=1;

x[sa[0]]=0;

for (i=1; i<n; i++)

x[sa[i]]=(y[sa[i-1]]==y[sa[i]] && y[sa[i-1]+k]==y[sa[i]+k])? p-1 : p++;

if (p>=n) break;

m=p;

}

}

int d[maxn][22];

void RMQ\_init()

{

for (int i=1; i<=n; i++) d[i][0]=height[i];

for (int j=1; (1<<j)<=n; j++)

for (int i=1; (i+(1<<j)-1)<=n; i++)

d[i][j]=min(d[i][j-1],d[i+(1<<(j-1))][j-1]);

}

int RMQ(int L,int R)

{

if (L>R) swap(L,R);

L++;

int k=0;

while((1<<(k+1))<=R-L+1) k++;

return min(d[L][k],d[R-(1<<k)+1][k]);

}

int main(){

int tt=0;

scanf("%d",&tt);

while(tt--)

{

scanf("%d",&n);

memset(s1,0,sizeof s1);

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

scanf("%s",&s1[i]);

n=strlen(s1);

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

s[i]=idx(s1[i]);

s[n]=0;

build\_ss(4,n);

getheight(n);

RMQ\_init();

int ans=1,len=0;

if (n==0) ans=0;

for (int l=1; l<=n/2; l++)

{

for (int i=0; (i+1)\*l<n; i++)

{

int p1=i\*l,p2=(i+1)\*l;

int k=RMQ(rank[p1],rank[p2]);

int r=l-k%l;

int p3=p1-r;

if (p3>=0 && k%l && RMQ(rank[p3],rank[p3+l])>=k) k+=r;

int c=k/l+1;

ans=max(ans,c);

}

}

cout<<ans<<endl;

}

}

例9：最长公共子串(pku2774,ural1517)

**[PKU2774(Long Long Message)求最长公共子串(后缀数组)](http://blog.csdn.net/jarily/article/details/8709964)**

[POI 2000 Repetitions 最长公共子串](http://blog.csdn.net/self_chou/article/details/7761180)

**[PKU2774(Long Long Message)求最长公共子串(后缀数组)](http://blog.csdn.net/jarily/article/details/8709964)**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

题目大意：

给定两个字符串A和B，求最长公共子串;

算法思想：

字符串的任何一个子串都是这个字符串的某个后缀的前缀;

求A和B的最长公共子串等价于求A的后缀和B的后缀的最长公共前缀的最大值;

如果枚举A和B的所有的后缀，很明显效率低下;

由于要计算A的后缀和B的后缀的最长公共前缀，所以先将第二个字符串写在第一个字符串后面，

中间用一个没有出现过的字符隔开，再求这个新的字符串的后缀数组;

后缀是指从某个位置i开始到整个串末尾结束的一个特殊子串;

字符串r的从第i个字符开始的后缀表示为Suffix(i),也就是Suffix(i)=r[i...len(r)];

后缀数组保存的是一个字符串的所有后缀的排序结果;

其中SA[i]保存的是字符串所有的后缀中第i小的后缀的开头位置;

定义height[i]=suffix(sa[i-1])和suffix(sa[i])的最长公共前缀，

也就是排名相邻的两个后缀的最长公共前缀;

只有当suffix(sa[i-1])和suffix(sa[i])不是同一个字符串中的两个后缀时，

height[i]才是满足条件的,即满足条件的height数组的最大值就是这两个字符串的最长公共子串;

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<iostream>

#include<cstring>

#include<cstdlib>

#include<cstdio>

#include<climits>

#include<algorithm>

**using** **namespace** std;

**const** **int** N=200010;

**const** **int** M=128;//r数组的最大值

#define F(x) ((x)/3+((x)%3==1?0:tb))

#define G(x) ((x)<tb?(x)\*3+1:((x)-tb)\*3+2)

**int** wa[N],wb[N],wv[N],\_ws[M];

**int** c0(**int** \*r,**int** a,**int** b)

{

**return** r[a]==r[b]&&r[a+1]==r[b+1]&&r[a+2]==r[b+2];

}

**int** c12(**int** k,**int** \*r,**int** a,**int** b)

{

**if**(k==2)

**return** r[a]<r[b]||r[a]==r[b]&&c12(1,r,a+1,b+1);

**else**

**return** r[a]<r[b]||r[a]==r[b]&&wv[a+1]<wv[b+1];

}

**void** sort(**int** \*r,**int** \*a,**int** \*b,**int** n,**int** m)

{

**for**(**int** i=0; i<n; i++)

wv[i]=r[a[i]];

**for**(**int** i=0; i<m; i++)

\_ws[i]=0;

**for**(**int** i=0; i<n; i++)

\_ws[wv[i]]++;

**for**(**int** i=1; i<m; i++)

\_ws[i]+=\_ws[i-1];

**for**(**int** i=n-1; i>=0; i--)

b[--\_ws[wv[i]]]=a[i];

**return**;

}

**void** dc3(**int** \*r,**int** \*sa,**int** n,**int** m)

{

**int** \*rn=r+n,\*san=sa+n,ta=0,tb=(n+1)/3,tbc=0,p;

r[n]=r[n+1]=0;

**for**(**int** i=0; i<n; i++)

{

**if**(i%3!=0)

wa[tbc++]=i;

}

sort(r+2,wa,wb,tbc,m);

sort(r+1,wb,wa,tbc,m);

sort(r,wa,wb,tbc,m);

p=1,rn[F(wb[0])]=0;

**for**(**int** i=1; i<tbc; i++)

{

rn[F(wb[i])]=c0(r,wb[i-1],wb[i])?p-1:p++;

}

**if**(p<tbc)

dc3(rn,san,tbc,p);

**else**

**for**(**int** i=0; i<tbc; i++)

san[rn[i]]=i;

**for**(**int** i=0; i<tbc; i++)

{

**if**(san[i]<tb)

wb[ta++]=san[i]\*3;

}

**if**(n%3==1)

wb[ta++]=n-1;

sort(r,wb,wa,ta,m);

**for**(**int** i=0; i<tbc; i++)

wv[wb[i]=G(san[i])]=i;

**int** i,j;

**for**(i=0,j=0,p=0; i<ta && j<tbc; p++)

{

sa[p]=c12(wb[j]%3,r,wa[i],wb[j])?wa[i++]:wb[j++];

}

**for**(; i<ta; p++)

sa[p]=wa[i++];

**for**(; j<tbc; p++)

sa[p]=wb[j++];

**return**;

}

**int** rank[N],height[N];

**void** calheight(**int** \*r,**int** \*sa,**int** n)

{

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

**for**(**int** i=1; i<=n; i++)

rank[sa[i]]=i;

**for**(**int** i=0; i<n; height[rank[i++]]=k)

{

**for**(k?k--:0,j=sa[rank[i]-1]; r[i+k]==r[j+k]; k++);

}

**return**;

}

**char** s[N];

**int** r[N\*3],sa[N\*3];

**int** main()

{

//freopen("C:\\Users\\Administrator\\Desktop\\kd.txt","r",stdin);

**while**(~scanf("%s",s))

{

**int** j=strlen(s);

s[j]=1;

scanf("%s",s+j+1);

**int** n=strlen(s);

**for**(**int** i=0; i<n; i++)

r[i]=s[i];

r[n]=0;

dc3(r,sa,n+1,M);//128:字符串中最大的字符+1

calheight(r,sa,n);

**int** ans=0;

**for**(**int** i=2; i<=n; i++)

{

**if**(height[i]>ans)

{

**if**((j<sa[i-1] && j>sa[i])|| (j>sa[i-1] && j<sa[i]))//不是同一字符串

ans=height[i];

}

}

printf("%d\n",ans);

}

**return** 0;

}

**例10:长度不小于k的公共子串的个数(pku3415)**

**[POJ 3415 Common Substrings(长度不小于k 的公共子串的个数--后缀数组+单调栈优化)](http://m.blog.csdn.net/blog/kalilili/44786949)**

题意：给定两个字符串A 和B，求长度不小于k的公共子串的个数（可以相同）。  
样例1:  
A=“xx”，B=“xx”，k=1，长度不小于k 的公共子串的个数是5。  
样例2:

A =“aababaa”，B =“abaabaa”，k=2，长度不小于k 的公共子串的个数是22。

思路：

这道题需要对height数组分组后，用单调栈优化。对于LCP=L>K的前缀，对答案的贡献是L-K+1.即长度为K,K+1.....L的公共字串。对于每一组，栈里维护height值递增，这样保证了每个height的贡献量为height[i]-K+1,因为有定理LCP（i,j）=min（height[i+1],...height[j]）如果在height[i]之后插入一个height[k]<height[i],那么height[i]就等于废了，他的价值仅为height[k]了，对于之后的j，LCP（i,j）起码要<=height[k],像这种递减的值插入栈前，需要将栈里>=插入值的变成插入值，然后将插入值的宽度增加。因为这当中为了降低复杂度，用了一个sum来维护当前栈里的后缀与将要入栈的后缀的公共字串的个数。因此实际上的{将栈里>=插入值的变成插入值}的操作，体现在sum上，是对sum减去一段height之差\*宽度。

补充：因为rank=i的后缀能与其rank前后的后缀都可能产生公共子串，但此算法只维护其前面的，因此排在A串后缀后面的B串后缀，应该用扫描B串后缀来维护，因为B在A后，以B为参考系，A就在前，这就是为什么要分别扫A和B串的原因。

补充完这段，接着开始：比如我们正在扫A串，那么当B串入栈时，sum均会加上L-K+1，而之后当有height值小的要入栈，此时可以是A串的后缀，他虽然不增加sum的值，但它也起到了阻断作用，它是之前栈里height值>=它的值变废了，因为后面的后缀与栈里算LCP值时，起码要<=这个值，由于栈里原先的元素都加了L-K+1，假设现在入栈的height值为T（T<L），那么它对答案的贡献是T-K+1，它使原先栈里大于等于它废成T，贡献量变为T-K+1，相当于减少了L-K+1-（T-K+1）=L-K（高度差），因为之前入栈时已加在sum上，此时就要从sum中减去了，当然不能忘了乘宽度，因为这些height都会合并，他代表的是一系列height相同的元素，是有数目的。

对于每一个新的后缀，如果heigh值大于栈顶元素:

Sa[1]:ab

Sa[2]:abc

Sa[3]:abcd

Sa[4]:abcde

如果出现height值小于栈顶元素值:

Sa[1]:a

Sa[2]:ab

Sa[3]:abc

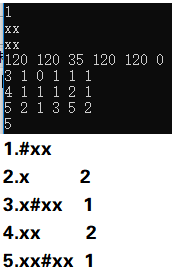
Sa[4]:abcd

Sa[5]:abcde

Sa[6]:abcdef

Sa[7]:abc

Sa[8]:ab



单调栈也是第一次遇到，其实就是维护一个栈里面的元素始终单调...  
[http://blog.csdn.net/hopeztm/article/details/7868581](http://blog.csdn.net/hopeztm/article/details/7868581" \t "_blank)这里有一个单调栈在一个经典DP中的运用

#include<iostream>

#include <cstdio>

#include <cstdlib>

#include <cstring>

#include <string>

#include <queue>

#include <algorithm>

#include <map>

#include <iomanip>

#define INF 99999999

typedef long long LL;

using namespace std;

const int MAX=2\*100000+10;

int \*Rank,r[MAX],sa[MAX],height[MAX];

int wa[MAX],wb[MAX],wm[MAX];

char s[MAX];

bool cmp(int \*r,int a,int b,int l){

return r[a] == r[b] && r[a+l] == r[b+l];

}

void makesa(int \*r,int \*sa,int n,int m){

int \*x=wa,\*y=wb,\*t;

for(int i=0;i<m;++i)wm[i]=0;

for(int i=0;i<n;++i)wm[x[i]=r[i]]++;

for(int i=1;i<m;++i)wm[i]+=wm[i-1];

for(int i=n-1;i>=0;--i)sa[--wm[x[i]]]=i;

for(int i=0,j=1,p=0;p<n;j=j\*2,m=p){

for(p=0,i=n-j;i<n;++i)y[p++]=i;

for(i=0;i<n;++i)if(sa[i]>=j)y[p++]=sa[i]-j;

for(i=0;i<m;++i)wm[i]=0;

for(i=0;i<n;++i)wm[x[y[i]]]++;

for(i=1;i<m;++i)wm[i]+=wm[i-1];

for(i=n-1;i>=0;--i)sa[--wm[x[y[i]]]]=y[i];

for(t=x,x=y,y=t,i=p=1,x[sa[0]]=0;i<n;++i){

x[sa[i]]=cmp(y,sa[i],sa[i-1],j)?p-1:p++;

}

}

Rank=x; //rank的问题

}

void calheight(int \*r,int \*sa,int n){

for(int i=0,j=0,k=0;i<n;height[Rank[i++]]=k){

for(k?--k:0,j=sa[Rank[i]-1];r[i+k] == r[j+k];++k);

}

}

LL calculate(int n,int len,int k){

int \*mark=wb,\*ans=wm,Top=0;//num[1],num[2]分别表示字符串A,B,suffix(0~i-1)和suffix(i)的最长公共子串>=k的总个数

LL sum=0,num[3]={0};

bool fl=true;

for(int i=1;i<=n;++i){

if(height[i]<k){

Top=num[1]=num[2]=0;

}else{

for(int size=Top;size && ans[size]>height[i]-k+1;--size){

//维护单调栈,ans记录的是suffix(j)和suffix(i-1)>=k的最长公共子串的个数,个数越多表示height[j]越大

num[mark[size]]+=height[i]-k+1-ans[size];

//suffix(j)和suffix(i)>=k的最长公共子串只能是长度为k~height[i],所以需要减去重复部分(ans[size]-(height[i]-k+1))

ans[size]=height[i]-k+1;//更新个数(更新单调栈,使栈里面元素非递减)

}

ans[++Top]=height[i]-k+1;

if(sa[i-1]<len)mark[Top]=1;//由于num新增加的结果是suffix(i-1)和suffix(i)的结果,所以是判断sa[i-1]

if(sa[i-1]>len)mark[Top]=2;

num[mark[Top]]+=height[i]-k+1;//增加由suffix(i-1)和suffix(i)产生的结果

if(sa[i]<len)sum+=num[2];//表示和suffix(i)产生的结果新增加B串的suffix(0~i-1)和suffix(i)>=k的个数

if(sa[i]>len)sum+=num[1];//表示和suffix(i)产生的结果新增加A串的suffix(0~i-1)和suffix(i)>=k的个数

//cout<<i<<" "<<height[i]<<" "<<num[1]<<" "<<num[2]<<" "<<sum<<" "<<height[i]<<endl;

}

}

return sum;

}

int main(){

int k,n,len;

while(~scanf("%d",&k),k){

scanf("%s",s);

for(n=0;s[n] != '\0';++n)r[n]=s[n];

r[len=n]='#';

scanf("%s",s+n+1);

for(++n;s[n] != '\0';++n)r[n]=s[n];

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

cout<<r[i]<<" ";cout<<endl;

r[n]=0;

makesa(r,sa,n+1,256);

calheight(r,sa,n);

cout<<calculate(n,len,k)<<endl;

}

return 0;

}

**例11:不小于k个字符串中的最长子串([pku3294](http://blog.sina.com.cn/s/blog_6635898a0102e0oi.html))**

[POJ 3294 Life Forms(不小于k/2个字符串中的最长子串 后缀数组)](http://blog.csdn.net/acm_cxlove/article/details/7947691)

<http://poj.org/problem?id=3294>

给定n个字符串，求出现在不小于k/2个字符串中的最长子串。

将n个字符串连起来，中间用不相同的且没有出现在字符串中的字符隔开， 求后缀数组。

然后二分答案，将后缀分成若干组，判断每组的后缀是否出现在不小于k个的原串中。这个做法的时间复杂度为O(nlogn)。

#include<iostream>

#include<cstdio>

#include<cstring>

#include<algorithm>

#include<cmath>

#define N 100005

#define LL long long

#define maxn 200005

**using** **namespace** std;

//以下为倍增算法求后缀数组

**int** wa[maxn],wb[maxn],wv[maxn],Ws[maxn];

**int** cmp(**int** \*r,**int** a,**int** b,**int** l)

{**return** r[a]==r[b]&&r[a+l]==r[b+l];}

**void** da(**const** **int** \*r,**int** \*sa,**int** n,**int** m){

**int** i,j,p,\*x=wa,\*y=wb,\*t;

**for**(i=0;i<m;i++) Ws[i]=0;

**for**(i=0;i<n;i++) Ws[x[i]=r[i]]++;

**for**(i=1;i<m;i++) Ws[i]+=Ws[i-1];

**for**(i=n-1;i>=0;i--) sa[--Ws[x[i]]]=i;

**for**(j=1,p=1;p<n;j\*=2,m=p){

**for**(p=0,i=n-j;i<n;i++) y[p++]=i;

**for**(i=0;i<n;i++) **if**(sa[i]>=j) y[p++]=sa[i]-j;

**for**(i=0;i<n;i++) wv[i]=x[y[i]];

**for**(i=0;i<m;i++) Ws[i]=0;

**for**(i=0;i<n;i++) Ws[wv[i]]++;

**for**(i=1;i<m;i++) Ws[i]+=Ws[i-1];

**for**(i=n-1;i>=0;i--) sa[--Ws[wv[i]]]=y[i];

**for**(t=x,x=y,y=t,p=1,x[sa[0]]=0,i=1;i<n;i++)

x[sa[i]]=cmp(y,sa[i-1],sa[i],j)?p-1:p++;

}

**return**;

}

**int** sa[maxn],Rank[maxn],height[maxn];

//求height数组

**void** calheight(**const** **int** \*r,**int** \*sa,**int** n){

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

**for**(i=1;i<=n;i++) Rank[sa[i]]=i;

**for**(i=0;i<n;height[Rank[i++]]=k)

**for**(k?k--:0,j=sa[Rank[i]-1];r[i+k]==r[j+k];k++);

**return**;

}

**char** ch[maxn];

**int** str[maxn];

**int** l[105];

**int** in[maxn];

**int** k;

**bool** check(**int** mid,**int** n,**int** out){

**bool** flag[105];

**int** cnt=0;

memset(flag,**false**,**sizeof**(flag));

**for**(**int** i=1;i<=n;i++){

**if**(height[i]<mid) {

**if**(cnt>k/2&&out){

**for**(**int** j=0,r=sa[i-1];j<mid;j++)

printf("%c",str[j+r]+'a'-1);

printf("\n");

}

cnt=0;

memset(flag,**false**,**sizeof**(flag));

}

**else**{

**if**(!flag[in[sa[i-1]]]){

flag[in[sa[i-1]]]=**true**;

cnt++;

}

**if**(!flag[in[sa[i]]]){

flag[in[sa[i]]]=**true**;

cnt++;

}

**if**(!out&&cnt>k/2) **return** **true**;

}

}

**return** **false**;

}

**int** main(){

**int** cnt=0;

**while**(scanf("%d",&k)!=EOF&&k){

**int** n=0;

**if**(cnt) printf("\n");

cnt++;

**for**(**int** i=0;i<k;i++){

scanf("%s",ch);

l[i]=strlen(ch);

**for**(**int** j=n;j<n+l[i];j++){

str[j]=ch[j-n]-'a'+1;

in[j]=i;

}

n+=l[i]+1;

str[n-1]=27+i;

}

n--;

str[n]=0;

da(str,sa,n+1,27+k+5);

calheight(str,sa,n);

**int** low=0,high=1000,mid,ans=-1;

**while**(low<=high){

mid=(low+high)/2;

**if**(check(mid,n,0)){ans=mid;low=mid+1;}

**else** high=mid-1;

}

**if**(ans<=0) puts("?");

**else** check(ans,n,1);

}

**return** 0;

}

例12

[SPOJ 220 Relevant Phrases of Annihilation（每个字符串至少出现两次且不重叠的最长子串）](http://blog.csdn.net/acm_cxlove/article/details/7947917)

题目：给定n个字符串，求出每个字符串至少出现两次且不重叠的最长子串

<http://www.spoj.pl/problems/PHRASES/>

将n个字符串连接起来，中间用不同的特殊字符隔开。求出后缀数组

二分答案，通过height值将后缀分组，判断在每一组中，每个字符串是否至少出现两次

而且由于题目需要不重叠的子串，所以还要记录每个字符串位置的最大值和最小值

判断其差是否小于当前答案

#include<iostream>

#include<cstdio>

#include<cstring>

#include<algorithm>

#include<cmath>

#define N 100005

#define LL long long

#define maxn 200005

#define inf 1<<30

**using** **namespace** std;

//以下为倍增算法求后缀数组

**int** wa[maxn],wb[maxn],wv[maxn],Ws[maxn];

**int** cmp(**int** \*r,**int** a,**int** b,**int** l)

{**return** r[a]==r[b]&&r[a+l]==r[b+l];}

**void** da(**const** **int** \*r,**int** \*sa,**int** n,**int** m){

**int** i,j,p,\*x=wa,\*y=wb,\*t;

**for**(i=0;i<m;i++) Ws[i]=0;

**for**(i=0;i<n;i++) Ws[x[i]=r[i]]++;

**for**(i=1;i<m;i++) Ws[i]+=Ws[i-1];

**for**(i=n-1;i>=0;i--) sa[--Ws[x[i]]]=i;

**for**(j=1,p=1;p<n;j\*=2,m=p){

**for**(p=0,i=n-j;i<n;i++) y[p++]=i;

**for**(i=0;i<n;i++) **if**(sa[i]>=j) y[p++]=sa[i]-j;

**for**(i=0;i<n;i++) wv[i]=x[y[i]];

**for**(i=0;i<m;i++) Ws[i]=0;

**for**(i=0;i<n;i++) Ws[wv[i]]++;

**for**(i=1;i<m;i++) Ws[i]+=Ws[i-1];

**for**(i=n-1;i>=0;i--) sa[--Ws[wv[i]]]=y[i];

**for**(t=x,x=y,y=t,p=1,x[sa[0]]=0,i=1;i<n;i++)

x[sa[i]]=cmp(y,sa[i-1],sa[i],j)?p-1:p++;

}

**return**;

}

**int** sa[maxn],Rank[maxn],height[maxn];

//求height数组

**void** calheight(**const** **int** \*r,**int** \*sa,**int** n){

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

**for**(i=1;i<=n;i++) Rank[sa[i]]=i;

**for**(i=0;i<n;height[Rank[i++]]=k)

**for**(k?k--:0,j=sa[Rank[i]-1];r[i+k]==r[j+k];k++);

**return**;

}

**char** ch[maxn];

**int** str[maxn];

**int** l[105];

**int** mx[15],mn[15];

**int** in[maxn];

**int** k;

**bool** check(**int** mid,**int** n){

**for**(**int** i=0;i<k;i++){mx[i]=0;mn[i]=inf;}

**for**(**int** i=1;i<=n;i++){

**if**(height[i]<mid){

**for**(**int** j=0;j<k;j++){

mx[j]=0;

mn[j]=inf;

}

mx[in[sa[i]]]=sa[i];

mn[in[sa[i]]]=sa[i];

}

**else**{

mx[in[sa[i]]]=max(mx[in[sa[i]]],sa[i]);

mn[in[sa[i]]]=min(mn[in[sa[i]]],sa[i]);

mx[in[sa[i-1]]]=max(mx[in[sa[i-1]]],sa[i-1]);

mn[in[sa[i-1]]]=min(mn[in[sa[i-1]]],sa[i-1]);

**int** j;

**for**(j=0;j<k;j++){

**if**(mx[j]-mn[j]<mid)

**break**;

}

**if**(j==k) **return** **true**;

}

}

**return** **false**;

}

**int** main(){

**int** cnt=0,t;

scanf("%d",&t);

**while**(t--){

scanf("%d",&k);

**int** n=0;

**for**(**int** i=0;i<k;i++){

scanf("%s",ch);

l[i]=strlen(ch);

**for**(**int** j=n;j<n+l[i];j++){

str[j]=ch[j-n]-'a'+1;

in[j]=i;

}

n+=l[i]+1;

str[n-1]=27+i;

}

n--;

str[n]=0;

da(str,sa,n+1,27+k+5);

calheight(str,sa,n);

**int** low=0,high=10000,mid,ans=0;

**while**(low<=high){

mid=(low+high)/2;

**if**(check(mid,n)){ans=mid;low=mid+1;}

**else** high=mid-1;

}

printf("%d\n",ans);

}

**return** 0;

}