# [【后缀自动机】](http://www.cnblogs.com/Lyush/p/3281546.html)

后缀自动机也是解决字符串问题的常用工具，犀利在O(N)的空间复杂度下存在给定串的后缀以及子串，而且支持在线的操作。

POJ-1509 [Glass Beads](http://poj.org/problem?id=1509)

**题意：求一个字符串的最小表示的开始下标。**

分析：其实有一个O(N)的算法专门来解决这个问题，并且实现非常简单，不过后缀自动机同样能够解决这个问题。首先把这个串重复两次，然后从前往后一一将字符加入到后缀自动机中，最后从根开始向下遍历串的长度层即可。

#include <cstdlib>

#include <cstring>

#include <cstdio>

#include <algorithm>

using namespace std;

const int N = 10005;

char str[N];

struct SAM{

struct Node{

int ch[26];

int f, len;

void init(){

f = -1, len = 0;

memset(ch, 0xff, sizeof (ch));

}

};

Node sn[N<<1];

int idx, last;

void init(){

idx = last = 0;

sn[idx++].init();

}

int newnode(){

sn[idx].init();

return idx++;

}

void add(int c){

int end = newnode();

int tmp = last;

sn[end].len = sn[last].len + 1;

for ( ; tmp != -1 && sn[tmp].ch[c] == -1; tmp = sn[tmp].f){

sn[tmp].ch[c] = end;

}

if (tmp == -1) sn[end].f = 0; // 所有的上一轮可接受点都没有指向字符c的孩子节点

else{

int nxt = sn[tmp].ch[c];

if (sn[tmp].len + 1 == sn[nxt].len) sn[end].f = nxt; // 如果可接受点有向c的转移，且长度只加1，那么该孩子可以替代当前的end，并且end的双亲指向该孩子

else{

int np = newnode();

sn[np] = sn[nxt];

sn[np].len = sn[tmp].len + 1;

sn[end].f = sn[nxt].f = np;

for (; tmp != -1 && sn[tmp].ch[c] == nxt; tmp = sn[tmp].f){

sn[tmp].ch[c] = np;

}

}

}

last = end;

}

};

SAM sam;

int main(){

int T;

scanf("%d", &T);

while (T--){

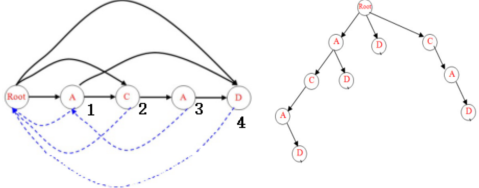
sam.init();

scanf("%s", str);

int len = strlen(str);

for (int i = 0; i < len\*2; ++i){

sam.add(str[i%len]-'a');

 }

int p = 0;

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

for (int j = 0; j < 26; ++j){

if (sam.sn[p].ch[j] != -1){

p = sam.sn[p].ch[j];

break;

}

}

}

printf("%d\n", sam.sn[p].len-len+1);

}

return 0;

}

SPOJ-1811 **[Longest Common Substring](http://www.spoj.com/problems/LCS/)**

**题意：求两个串的最长公共子串。**

分析：先用第一个串构造出后缀自动机，然后逐个的匹配第二个串，如果当前节点失配，那么找 f 节点。

#include <cstdlib>

#include <cstring>

#include <cstdio>

#include <algorithm>

using namespace std;

const int N = 250010;

char s1[N], s2[N];

struct SAM{

struct{

int len, f, ch[26];

void init(){

len = 0, f = -1;

memset(ch, 0xff, sizeof (ch));

}

} e[N<<1];

int idx, last;

void init(){

idx = last = 0;

e[idx++].init();

}

int newnode(){

e[idx].init();

return idx++;

}

void add(int c){

int end = newnode();

int tmp = last;

e[end].len = e[last].len + 1;

for (; tmp != -1 && e[tmp].ch[c] == -1; tmp = e[tmp].f){

e[tmp].ch[c] = end;

}

if (tmp == -1) e[end].f = 0;

else{

int nxt = e[tmp].ch[c];

if (e[tmp].len + 1 == e[nxt].len) e[end].f = nxt;

else{

int np = newnode();

e[np] = e[nxt];

e[np].len = e[tmp].len + 1;

e[nxt].f = e[end].f = np;

for (; tmp != -1 && e[tmp].ch[c] == nxt; tmp = e[tmp].f){

e[tmp].ch[c] = np;

}

}

}

last = end;

}

};

SAM sam;

int main(){

while (scanf("%s %s", s1, s2) != EOF){

sam.init();

int len1 = strlen(s1);

int len2 = strlen(s2);

for (int i = 0; i < len1; ++i){ // 构造好第一个字符串的后缀自动机

sam.add(s1[i]-'a');

}

int p = 0, ret = 0, clen = 0;

for (int i = 0; i < len2; ++i){

int id = s2[i]-'a';

if (sam.e[p].ch[id] != -1){

clen++;

p = sam.e[p].ch[id];

} else{

for(; p != -1 && sam.e[p].ch[id] == -1; p = sam.e[p].f) ;

if (p == -1) clen = 0, p = 0;

else{

clen = sam.e[p].len + 1;

p = sam.e[p].ch[id];

}

}

ret = max(clen, ret);

}

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

}

return 0;

}

SPOJ-8222 **[Substrings](http://www.spoj.com/problems/NSUBSTR/)**

# **「SPOJ8222」Substrings**

You are given a string S which consists of 250000 lowercase latin letters at most. We define F(x) as the maximal number of times that some string with length x appears in S. For example for string ‘ababa’ F(3) will be 2 because there is a string ‘aba’ that occurs twice. Your task is to output F(i) for every i so that 1<=i<=|S|.

### Input

String S consists of at most 250000 lowercase latin letters.

### Output

Output |S| lines. On the i-th line output F(i).

### Example

Input:

ababa

Output:

3

2

2

1

1

**题意：给定一个字符串，求出现次数最多的长度为 i 的子串的次数，i 的取值为 1 - len，len表示这个字符串的长度。**

这是一道SAM的入门题。

我们首先构造出SAM，然后SAM中每个节点i有一个righti和leni，那么长度在(lenfai,leni]内的都出现了righti次。考虑如果一个长度为leni的出现了x次，那么长度<leni的一定也至少出现了x次，所以我们可以直接算每个righti对leni的贡献。最后递推更新一下答案令fi=max(fi,fi+1)就好了。

现在还有一个问题没解决，就是righti的求法，我们只需要righti的大小而不需要集合内究竟是哪些数，所以可以开始给每个代表s[1..i]的节点i的righti=1，然后按深度合并一下，就能O(|S|)内求出所有的right了。

#include<iostream>

#include<cstdio>

#include<cstring>

#include<algorithm>

#define N 500005

using namespace std;

char s[N];

int S,n,cnt,last;

int a[N],f[N],t[N],fa[N],cc[N],len[N],r[N],ch[N][26];

inline void insert(int x)

{

int c=a[x];

int p=last,np=++cnt; last=np;

len[np]=x;

while (p&&!ch[p][c]) ch[p][c]=np,p=fa[p];

if (!p) fa[np]=S;

else

{

int q=ch[p][c];

if (len[p]+1==len[q]) fa[np]=q;

else

{

int nq=++cnt;

len[nq]=len[p]+1;

memcpy(ch[nq],ch[q],sizeof(ch[q]));

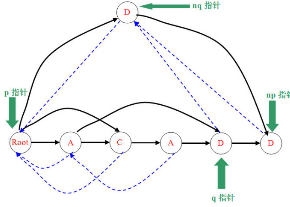
fa[nq]=fa[q]; fa[np]=fa[q]=nq;

while (ch[p][c]==q) ch[p][c]=nq,p=fa[p];

}

}

}

int main()

{

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

last=S=++cnt;

n=strlen(s+1);

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

a[i]=s[i]-'a';

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

insert(i);

for (int i=1,p=S;i<=n;i++)

p=ch[p][a[i]],r[p]++;//后缀子串出现1次

for (int i=1;i<=cnt;i++) ++cc[len[i]];//基数排序

for (int i=1;i<=n;i++) cc[i]+=cc[i-1];

for (int i=cnt;i>=1;i--) t[cc[len[i]]--]=i;

for (int i=cnt;i;i--) r[fa[t[i]]]+=r[t[i]];//后缀子串累加到父新节点

for (int i=1;i<=cnt;i++) f[len[i]]=max(f[len[i]],r[i]);//保存相同长度的最大值

for (int i=n;i;i--) f[i]=max(f[i],f[i+1]);短的子串包含在长的子串中。

for (int i=1;i<=n;i++) printf("%d\n",f[i]);

return 0;

}

spoj8222 Substrings

题意：f[x]表示所有长度为 x 的子串中，出现次数的最大值。求所有f[x]

方法：建立SAM，根据拓扑序找到长度为 x 的子串个数，更新一下就行了

#include <stdio.h>

#include <string.h>

#define N 250010

char str[N];

int root,cnt=0,n,last,ans[N],son[N<<1][26],mx[N<<1],size[N<<1],fa[N<<1],c[N<<1],a[N<<1];

inline int max(int x,int y){

if(x>y) return x;

return y;

}

inline void ins(int ch){

int p=last,np=++cnt;last=np;mx[np]=mx[p]+1;size[np]=1;

while(p && !son[p][ch]) son[p][ch]=np,p=fa[p];

if(!p) fa[np]=root;

else{

int q=son[p][ch];

if(mx[q]==mx[p]+1) fa[np]=q;

else{

int nq=++cnt;mx[nq]=mx[p]+1;

memcpy(son[nq],son[q],sizeof(son[q]));

fa[nq]=fa[q];fa[q]=fa[np]=nq;

while(son[p][ch]==q) son[p][ch]=nq,p=fa[p];

}

}

}

int main(){

scanf("%s",str+1);n=strlen(str+1);last=root=++cnt;

for(int i=1;i<=n;i++) ins(str[i]-'a');

for(int i=1;i<=cnt;i++) c[mx[i]]++;

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

for(int i=cnt;i>=1;i--) a[c[mx[i]]--]=i;

for(int i=cnt;i>=1;i--){

int p=a[i];size[fa[p]]+=size[p];

ans[mx[p]]=max(ans[mx[p]],size[p]);

}for(int i=1;i<=n;i++) printf("%d\n",ans[i]);

return 0;

}

SPOJ-1812 **[Longest Common Substring II](http://www.spoj.com/problems/LCS2/)**

**题意：求多个串的最长公共子串。**

分析：首先将第一个串建立一个后缀自动机，然后以该串为参照，分别对应每个串求出后缀自动机中的每个节点所能够匹配的最长子串长度，需要对每个节点按照len值排一个序，用以在包含子串之间进行更新。每更新完一轮值，其实就可以相应的缩小这个len值了，因为是所有串的公共子串，因此是由最小值来决定的。

本题容易看出就是分别将所有串的所有匹配长度记录在状态上，然后取所有串记录值的min，后再对所有状态取max。

但是不要忘记了一点：更新parent树的祖先。

为什么呢？首先如果子树被匹配过了，那么长度一定大于任意祖先匹配的长度（甚至有些祖先匹配长度为0！为什么呢，因为我们在匹配的过程中，只是找到一个子串，可能还遗漏了祖先没有匹配到，这样导致了祖先的记录值为0，那么在对对应状态取min的时候会取到0，这样就wa了。而且注意，如果匹配到了当前节点，那么祖先们一定都可以赋值为祖先的length！因为当前节点的length大于任意祖先。

比如数据

acbbc  
bc  
ac

答案应该是1没错吧。如果没有更新祖先，那么答案会成0。

这个多想想就行了。

所以以后记住：对任意多串匹配时，凡是对同一个状态取值时，要注意当前状态的子树是否比当前状态记录的值优。

时间复杂度：线性。

#include <cstdio>

#include <cstring>

#include <cstdlib>

#include <algorithm>

using namespace std;

const int N = 100005;

struct Node{

int len, f;

int ch[26];

void init(){

len = 0, f = -1;

memset(ch, 0xff, sizeof (ch));

}

};

struct SAM{

Node e[N<<1];

int idx, last;

int newnd(){

e[idx].init();

return idx++;

}

void init(){

idx = 0;

last = newnd();

}

void add(int c){

int end = newnd();

int tmp = last;

e[end].len = e[last].len + 1;

for (; tmp != -1 && e[tmp].ch[c] == -1; tmp = e[tmp].f){

e[tmp].ch[c] = end;

}

if (tmp == -1) e[end].f = 0;

else{

int nxt = e[tmp].ch[c];

if (e[tmp].len + 1 == e[nxt].len) e[end].f = nxt;

else{

int nd = newnd();

e[nd] = e[nxt];

e[nd].len = e[tmp].len + 1;

e[nxt].f = e[end].f = nd;

for (; tmp != -1 && e[tmp].ch[c] == nxt; tmp = e[tmp].f){

e[tmp].ch[c] = nd;

}

}

}

last = end;

}

};

char str[N];

SAM sam;

int ws[N];

int pk[N<<1];

int dp[N<<1]; // dp[i]表示第i号节点为终点能够匹配最长的子串长度

int main(){

sam.init();

scanf("%s", str);

int slen = strlen(str);

for (int i = 0; i < slen; ++i) sam.add(str[i] - 'a');

// 按照能够匹配的子串长度按照从大到小排序

for (int i = 0; i <= slen; ++i) ws[i] = 0;

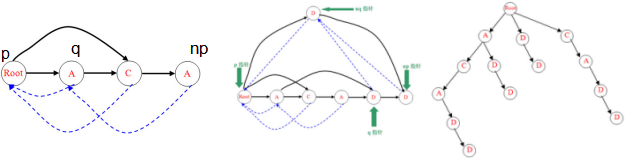
for (int i = 1; i < sam.idx; ++i) ws[sam.e[i].len]++;

for (int i = 1; i <= slen; ++i) ws[i] += ws[i-1];

for (int i = sam.idx-1; i > 0; --i) pk[ws[sam.e[i].len]--] = i;

Node \*ele = sam.e;

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

 slen = strlen(str);

int len = 0;

for (int i = 0, p = 0; i < slen; ++i){

int c = str[i] - 'a';

if (ele[p].ch[c] != -1){

++len;

p = ele[p].ch[c];

dp[p] = max(len, dp[p]);

} else{

while (p != -1 && ele[p].ch[c] == -1) p = ele[p].f;

if (p != -1){

len = ele[p].len + 1;

p = ele[p].ch[c];

dp[p] = max(len, dp[p]); // dp[i]表示第i号节点为终点能够匹配最长的子串长度

}

else len = p = 0;

}

}

for (int i = sam.idx-1; i > 0; --i){

int v = pk[i];

ele[v].len = min(ele[v].len, dp[v]); // 缩小能够匹配的最长子串长度

//dp[ele[v].f] = max(dp[ele[v].f], dp[v]); // 扩充包含子串的匹配长度

if (dp[v] && ele[v].f) dp[ele[v].f] = ele[ele[v].f].len; //与kmp一样，找最大匹配

dp[v] = 0;

}

}

int ans = 0;

for (int i = 1; i < sam.idx; ++i) ans = max(ans, ele[i].len);

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

return 0;

}

[spoj 7258 Lexicographical Substring Search (后缀自动机)](http://www.spoj.com/problems/SUBLEX/" \t "_blank)

题意：给出一个字符串，长度为90000。询问q次，每次回答一个k，求字典序第k小的子串。

解题思路：构造出sam后，类似splay求前驱的做法，不断的逼近答案。我们知道，sam里从s走到某一节点即为一个子串，所以我们在每个节点下记录一个cnt，表示该节点下，能走到的节点有多少个。那么我们在求第k小的子串时，就往下走，枚举当前节点下的26字母节点，若枚举到的节点的cnt+1>=k那么就往该节点走，并输出这条边上的字母（为什么要+1呢？因为走到这个节点就可以是一个子串）。否则k -= cnt[v] + 1 （+1的理由同上）。还有一个问题就是如何快速统计cnt了，这个留个小思考吧，其实方法前面几道题里都用到了，详细讨论请留言。

另外，这题还有个小优化，我们要把空的字母边缩掉，这个大家自己去发现了。我也因为这个T了好几发

#include<stdio.h>

#include<string.h>

#include<algorithm>

**using** **namespace** std;

**const** **int** maxn=90005;

**int** fa[maxn<<1],c[26][maxn<<1],val[maxn<<1];

**int** last,tot;

**int** cnt[maxn<<2];

**int** max(**int** a,**int** b){ **return** a>b ? a : b;}

**inline** **int** new\_node(**int** step){

**int** i;

    val[++tot]=step;

**for**(i=0;i<26;i++)c[i][tot]=0;

    fa[tot]=0;

**return** tot;

}

**void** add(**int** k){

**int** p=last,i;

**int** np=new\_node(val[p]+1);

**while**(p && !c[k][p])c[k][p]=np,p=fa[p];

**if**(!p)fa[np]=1;

**else** {

**int** q=c[k][p];

**if**(val[q] == val[p]+1)fa[np]=q;

**else** {

**int** nq=new\_node(val[p]+1);

**for**(i=0;i<26;i++)c[i][nq]=c[i][q];

            fa[nq]=fa[q];

            fa[np]=fa[q]=nq;

**while**(p && c[k][p] == q)c[k][p]=nq,p=fa[p];

        }

    }

    last=np;

}

**void** init(){

    tot=0;

    last=new\_node(0);

}

**char** s[maxn];

**int** pos[maxn<<1],ws[maxn<<1],to[maxn<<1];

**int** main(){

    scanf("%s",s);

    init ();

**int** i,len=strlen(s), j;

**for**(i=0;i<len;i++)add(s[i]-'a');

**for**(i=1;i<=tot;i++)ws[i]=0;

**for**(i=1;i<=tot;i++)ws[val[i]]++;

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

**for**(i=1;i<=tot;i++)pos[ws[val[i]]--]=i;

**for**(i=tot;i>=1;i--){

**int** p=pos[i];

**int** k=0;

**for**(j=0;j<26;j++){

**if**(c[j][p]){

                cnt[p]+=cnt[c[j][p]]+1;

                c[k++][p]=c[j][p]; //连续

                to[c[k-1][p]]=j+'a';

            }

        }

        c[k][p]=0;

    }

**int** q;

    scanf("%d",&q);

**while**(q--){

**int** k;

        scanf("%d",&k);

**int** p=1;

**while**(k>0){

            i=0;

**while**(c[i][p]){  //i为字母指针，p为节点

**int** r=c[i][p];

**if**(cnt[r]+1>=k){

                    printf("%c",to[r]);

                    k--;

                    p=r;

**break**;

                }

**else** k-=cnt[r]+1;

                i++;

            }

        }

        puts("");

    }

}

HDU-4622 [Reincarnation](http://acm.hdu.edu.cn/showproblem.php?pid=4622)

**题意：给定一个字符串，长度最长为2000，有至多10000组询问，每个询问给定一个区间，求出该区间内共有多少个不同的子串。**

分析：一开始是直接对每一个询问构建一个后缀自动机，超时了。正解是对询问进行排序，拥有相同L值的区间可以合并，由于最多只有2000个字符，因此重建的次数就最多就是2000次。对于给定的后缀自动机，子串的个数是遍历每一个节点，ans += ele[i].len - ele[ele[i].f].len，意思为枚举每个子串的最后一个元素，新增的子串个数就是到该点最长后缀减去与其父亲节点的重复后缀部分。后缀数组则是在遍历height数组的时候确定每个子串的开始位置，然后减去相同的前缀部分。

#include <cstdlib>

#include <cstdio>

#include <cstring> 

#include <algorithm>

using namespace std;

const int N = 2005;

char str[N];

struct Node{

int f, len, ch[26];

void init(){

len = 0, f = -1;

memset(ch, 0xff, sizeof (ch));

}

};

int pk[N<<1];

int ws[N];

int dp[N<<1];

struct SAM{

Node e[N<<1];

int idx, last;

void init(){

idx = 0;

last = newnd();

}

int newnd(){

e[idx].init();

return idx++;

}

void add(int c){

int end = newnd();

int p = last;

e[end].len = e[p].len + 1;

for (; p != -1 && e[p].ch[c] == -1; p = e[p].f){

e[p].ch[c] = end;

}

if (p == -1) e[end].f = 0;

else{

int nxt = e[p].ch[c];

if (e[p].len + 1 == e[nxt].len) e[end].f = nxt;

else{

int nd = newnd();

e[nd] = e[nxt];

e[nd].len = e[p].len + 1;

e[nxt].f = e[end].f = nd;

for (; p != -1 && e[p].ch[c] == nxt; p = e[p].f){

e[p].ch[c] = nd;

}

}

}

last = end;

}

};

SAM sam;

int cal(){

int ret = 0;

Node \*ele = sam.e;

for (int i = 1; i < sam.idx; ++i){

ret += ele[i].len - ele[ele[i].f].len;

}

return ret;

}

struct Query{

int l, r, No;

bool operator < (const Query & t) const{

if (l != t.l) return l < t.l;

else return r < t.r;

}

};

Query seq[10005];

int ans[10005];

void solve(){

int Q;

scanf("%d", &Q);

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

scanf("%d %d", &seq[i].l, &seq[i].r);

seq[i].No = i;

}

sort(seq+1, seq+1+Q);

seq[0].l = -1;

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

if (seq[i].l != seq[i-1].l){

sam.init();

for (int j = seq[i].l; j <= seq[i].r; ++j){

sam.add(str[j] - 'a');

}

ans[seq[i].No] = cal();

} else{

for (int j = seq[i-1].r + 1; j <= seq[i].r; ++j){

sam.add(str[j] - 'a');

}

ans[seq[i].No] = cal();

}

}

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

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

}

}

int main(){

int T;

scanf("%d", &T);

while (T--){

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

solve();

}

return 0;

}

HDU-4641 [K-string](http://acm.hdu.edu.cn/showproblem.php?pid=4641)

**题意：给定一个原始的字符串，有m次操作，每次操作可以向该字符串末尾添加一个字符或者询问在字符串中出现了至少K次的子串一共有多少个？**

分析：首先使用后缀自动机的这种写法在极端情况下（K非常大，字符串全为同一个字符）是会TLE的，不过该题能够使用该算法水过。在后缀自动机的节点中添加新的节点信息：num，表示以该节点结束的子串出现的次数。那么没插入一个元素就遍历 f 指针更新与当前后缀有共同后缀的更短的子串，如果次数已经等于K，说明已经被统计过了，立即退出遍历过程，因为更前面的也一定被统计过了；如果加一之后刚好等于K则说明这个子串可以被统计了，子串的个数为 e[tmp].len - e[e[tmp].f].len 统计后也立即退出，否则一直往前更新。

#include <cstdlib>

#include <cstring>

#include <cstdio>

#include <algorithm>

using namespace std;

typedef long long LL;

const int N = 50005;

const int M = 200005;

int n, K, m;

char str[N];

LL ans;

struct Node{

int len, f, num, ch[26];

void init(){

len = num = 0;

f = -1;

memset(ch, 0xff, sizeof (ch));

}

};

struct SAM{

Node e[N+M<<1];

int idx, last;

int newnd(){

e[idx].init();

return idx++;

}

void init(){

idx = 0;

last = newnd();

}

void add(int c){

// printf("\_\_%c\_\_\n", c + 'a');

int end = newnd();

int tmp = last;

e[end].len = e[last].len + 1;

for (; tmp != -1 && e[tmp].ch[c] == -1; tmp = e[tmp].f){

e[tmp].ch[c] = end;

}

if (tmp == -1) e[end].f = 0;

else{

int nxt = e[tmp].ch[c];

if (e[tmp].len+1 == e[nxt].len) e[end].f = nxt;

else{

int nd = newnd();

e[nd] = e[nxt];

e[nd].len = e[tmp].len + 1;

e[nxt].f = e[end].f = nd;

for (; tmp != -1 && e[tmp].ch[c] == nxt; tmp = e[tmp].f){

e[tmp].ch[c] = nd;

}

}

}

for (tmp = end; tmp != 0; tmp = e[tmp].f){

if (e[tmp].num == K) break;

e[tmp].num++;

if (e[tmp].num == K){

ans += e[tmp].len - e[e[tmp].f].len;

break;

}

}

last = end;

}

};

SAM sam;

int main(){

while (scanf("%d %d %d", &n, &m, &K) != EOF){

sam.init();

ans = 0;

scanf("%s", str);

int slen = strlen(str), op;

for (int i = 0; i < slen; ++i) sam.add(str[i] - 'a');

while (m--){

scanf("%d", &op);

if (op == 1){

scanf("%s", str);

sam.add(str[0] - 'a');

} else{

printf("%I64d\n", ans);

}

}

}

return 0;

}