/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*AC自动机 \*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <stdio.h>#include <stdlib.h>#include <string.h>

struct Node

{

int cnt;//是否为该单词的最后一个结点

Node \*fail;//失败指针

Node \*next[26];//Trie中每个结点的各个节点

}\*queue[500005];//队列，方便用BFS构造失败指针

char s[1000005];//主字符串

char keyword[55];//需要查找的单词

Node \*root;//头结点

void Init(Node \*root)//每个结点的初始化

{

root->cnt=0;

root->fail=NULL;

for(int i=0;i<26;i++)

root->next[i]=NULL;

}

void Build\_trie(char \*keyword)//构建Trie树

{

Node \*p,\*q;

int i,v;

int len=strlen(keyword);

for(i=0,p=root;i<len;i++)

{

v=keyword[i]-'a';

if(p->next[v]==NULL)

{

q=(struct Node \*)malloc(sizeof(Node));

Init(q);

p->next[v]=q;//结点链接

}

p=p->next[v];//指针移动到下一个结点

}

p->cnt++;//单词最后一个结点cnt++，代表一个单词

}

void Build\_AC\_automation(Node \*root)

{

int head=0,tail=0;//队列头、尾指针

queue[head++]=root;//先将root入队

while(head!=tail)

{

Node \*p=NULL;

Node \*temp=queue[tail++];//弹出队头结点

for(int i=0;i<26;i++)

{

if(temp->next[i]!=NULL)//找到实际存在的字符结点

{ //temp->next[i] 为该结点，temp为其父结点

if(temp==root)//若是第一层中的字符结点，则把该结点的失败指针指向root

temp->next[i]->fail=root;

else

{

//依次回溯该节点的父节点的失败指针直到某节点的next[i]与该节点相同，

//则把该节点的失败指针指向该next[i]节点；

//若回溯到 root 都没有找到，则该节点的失败指针指向 root

p=temp->fail;//将该结点的父结点的失败指针给p

while(p!=NULL)

{

if(p->next[i]!=NULL)

{

temp->next[i]->fail=p->next[i];

break;

}

p=p->fail;

}

//让该结点的失败指针也指向root

if(p==NULL)

temp->next[i]->fail=root;

}

queue[head++]=temp->next[i];//每处理一个结点，都让该结点的所有孩子依次入队

}

}

}

}

int query(Node \*root)

{ //i为主串指针，p为模式串指针

int i,v,count=0;

Node \*p=root;

int len=strlen(s);

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

{

v=s[i]-'a';

//由失败指针回溯查找，判断s[i]是否存在于Trie树中

while(p->next[v]==NULL && p!=root)

p=p->fail;

p=p->next[v];//找到后p指针指向该结点

if(p==NULL)//若指针返回为空，则没有找到与之匹配的字符

p=root;

Node \*temp=p;//匹配该结点后，沿其失败指针回溯，判断其它结点是否匹配

while(temp!=root)//匹配结束控制

{

if(temp->cnt>=0)//判断该结点是否被访问

{

count+=temp->cnt;//由于cnt初始化为 0，所以只有cnt>0时才统计了单词的个数

temp->cnt=-1;//标记已访问过

}

else//结点已访问，退出循环

break;

temp=temp->fail;//回溯 失败指针 继续寻找下一个满足条件的结点

}

}

return count;

}

int main()

{

int T,n;

scanf("%d",&T);

while(T--)

{

root=(struct Node \*)malloc(sizeof(Node));

Init(root);

scanf("%d",&n);

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

{

scanf("\n%s",keyword);

Build\_trie(keyword);

}

Build\_AC\_automation(root);

scanf("\n%s",s);

printf("%d\n",query(root));

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*AC自动机2 \*\*\*\*\*\*\*\*\*\*\*\*\*/

#include<bits/stdc++.h>

using namespace std;

const int maxn = 1e7 + 5;

const int MAX = 10000000;

struct node{

node \*next[26];

node \*fail;

int sum;

}\*root, \*q[MAX], \*newnode;

char key[70], pattern[maxn];

int head, tail, N, cnt;

void Insert(char \*s)

{

node \*p = root;

for(int i = 0; s[i]; i++)

{

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

if(p->next[x] == NULL)

{

newnode=(struct node \*)malloc(sizeof(struct node));

for(int j=0;j<26;j++) newnode->next[j] = 0;

newnode->sum = 0;newnode->fail = 0;

p->next[x]=newnode;

}

p = p->next[x];

}

p->sum++;

}

void build\_fail\_pointer()

{

head = 0;

tail = 1;

q[head] = root;

node \*p;

node \*temp;

while(head < tail)

{

temp = q[head++];

for(int i = 0; i <= 25; i++)

{

if(temp->next[i])

{

if(temp == root)

{

temp->next[i]->fail = root;

}

else

{

p = temp->fail;

while(p)

{

if(p->next[i])

{

temp->next[i]->fail = p->next[i];

break;

}

p = p->fail;

}

if(p == NULL) temp->next[i]->fail = root;

}

q[tail++] = temp->next[i];

}

}

}

}

void ac\_automation(char \*ch)

{

node \*p = root;

int len = strlen(ch);

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

{

int x = ch[i] - 'a';

while(!p->next[x] && p != root) p = p->fail;

p = p->next[x];

if(!p) p = root;

node \*temp = p;

while(temp != root)

{

if(temp->sum >= 0)

{

cnt += temp->sum;

temp->sum = -1;

}

else break;

temp = temp->fail;

}

}

}

int main()

{

int T;

scanf("%d",&T);

while(T--)

{

root=(struct node \*)malloc(sizeof(struct node));

for(int j=0;j<26;j++) root->next[j] = 0;

root->fail = 0;

root->sum = 0;

scanf("%d",&N);

getchar();

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

{

gets(key);

Insert(key);

}

gets(pattern);

cnt = 0;

build\_fail\_pointer();

ac\_automation(pattern);

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

}

return 0;

}