# 哈希散列

1. 基本术语
2. Hash方法：在存放记录时，通过相同函数计算存储位置，并按此位置存放，这种方法称为Hash方法。
3. Hash函数：是指在Hash方法中使用的函数。
4. Hash表：按Hash方法构造出来的表。
5. Hash地址：通过Hash函数计算记录的存储位置，我们把这个存储位置称为Hash地址
6. 冲突：不同记录的关键字经过Hash函数的计算可能是同一Hash地址，即key1≠key2时，H(key1)=H(key2),这种现象叫做冲突。

对于Hash方法需要讨论三个问题：

1. 装满因子
2. 对于给定的一个关键码集合，选择一个计算简单且地址分布比较均匀的Hash函数，避免或尽量减少冲突。
3. 拟定解决冲突的方案。
4. 装满因子

在散列存储中，若发生冲突，则必须采用特殊的方法来解决冲突问题，才能使散列查找能顺利进行。虽然冲突不可避免，但发生冲突的可能性却与三个方面因素有关。第一是与装填因子α有关，所谓装填因子是指散列表中已存入的元素个数n与散列表的大小m的比值，α=n/m。当α越小时，发生冲突的可能性越小，α越大（最大为1）时，发生冲突的可能性就越大。但是，为了减少冲突的发生，不能将α变得太小，这样将会造成大量存储空间的浪费，因此必须兼顾存储空间和冲突两个方面。第二是与所构造的散列函数有关。第三是与解决冲突的方法有关，这在后面会进一步介绍。

三、构造Hash函数考虑的因素有：

（1）计算哈希函数所需的时间

（2）关键字的长度

（3）哈希表的大小

（4）关键字的分布情况

（5）记录的查找频率

四、哈希函数比较常用的构造方法：

1. 直接定址法：

此类方法取记录中关键码的某个线性函数值作为

Hash地址：H(key)=a\*key+b (a,b为常数)

这类hash函数是一对一的映射，一般不会产生冲突。但是，它要求Hash地址间的大小与关键码集合的大小相同，这种要求一般很难实现。

这种方法的优点是：简单，均匀，不会产生冲突。但是需要事先知道关键字的分布情况，适合查找表较小并且连续的情况。

1. **除留余数法：**

设Hash表中允许的地址数为m，取一个不大于m，但最接近或等于m的质数p，或选取一个不含有小于20的质因子的合数作为除数。

Hash函数为：f(key)=key ％ p (p<=m)

其中：“％”是整数除法取余的运算，要求这时的质数p不是接近2的幂。

五、冲突处理方法

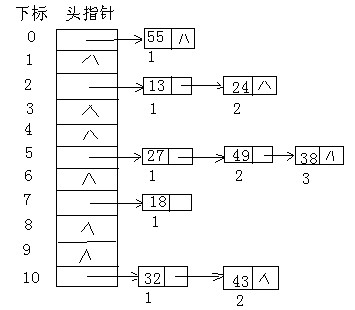
（1）开放定址法：

所谓开放地址法是指在冲突发生时，产生某个探测序列，按照这个序列，探测在Hash表中的其它存储单元直到探测到不发生冲突的存储单元为止。开放定址法的一般形式为：hi(key)=(H(key)+di)% m （i=1,2,…,m） 其中：Hi(key)为经过i次探测H(key)为关键字key的直接Hash地址，m为Hash表的长度，di为每次探测时的地址增量。

一般情况下，地址增量的取值常用的两种：

1. di=1，2，…，m-1 这种情况为线性探测再Hash;
2. di=12，-12，22，-22，…，±k2(k≤|m/2|) 称这种情况为二次探测再Hash.
3. 拉链法（链地址法）

链地址法是将通过Hash函数计算出来的Hash地址相同的关键码通过链表起来各链表表头节点组成一个向量。向量的元素个数与关键字个数相同。



与开放定址法相比，拉链法有如下几个优点：

①拉链法处理冲突简单，且无堆积现象，即非同义词决不会发生冲突，因此平均查找长度较短；

②由于拉链法中各链表上的结点空间是动态申请的，故它更适合于造表前无法确定表长的情况；

③开放定址法为减少冲突，要求装填因子α较小，故当结点规模较大时会浪费很多空间。而拉链法中可取α≥1，且结点较大时，拉链法中增加的指针域可忽略不计，因此节省空间；

理解线性探测法和拉链法例题：  
设散列函数为h(key)=key % 11；散列地址表空间为0~10，对关键字序列{27，13，55，32，18，49，24，38，43}，利用线性探测法解决冲突，构造散列表。  
　　解：首先根据散列函数计算散列地址：  
　　　　h(27)=5； h(13)=2；  
　　　　h(55)=0； h(32)=10；  
　　　　h(18)=7； h(49)=5；  
　　　　h(24)=2； h(38)=5；  
　　　　h(43)=10；　  
　　(散列表各元素查找比较次数标注在结点的上方或下方)

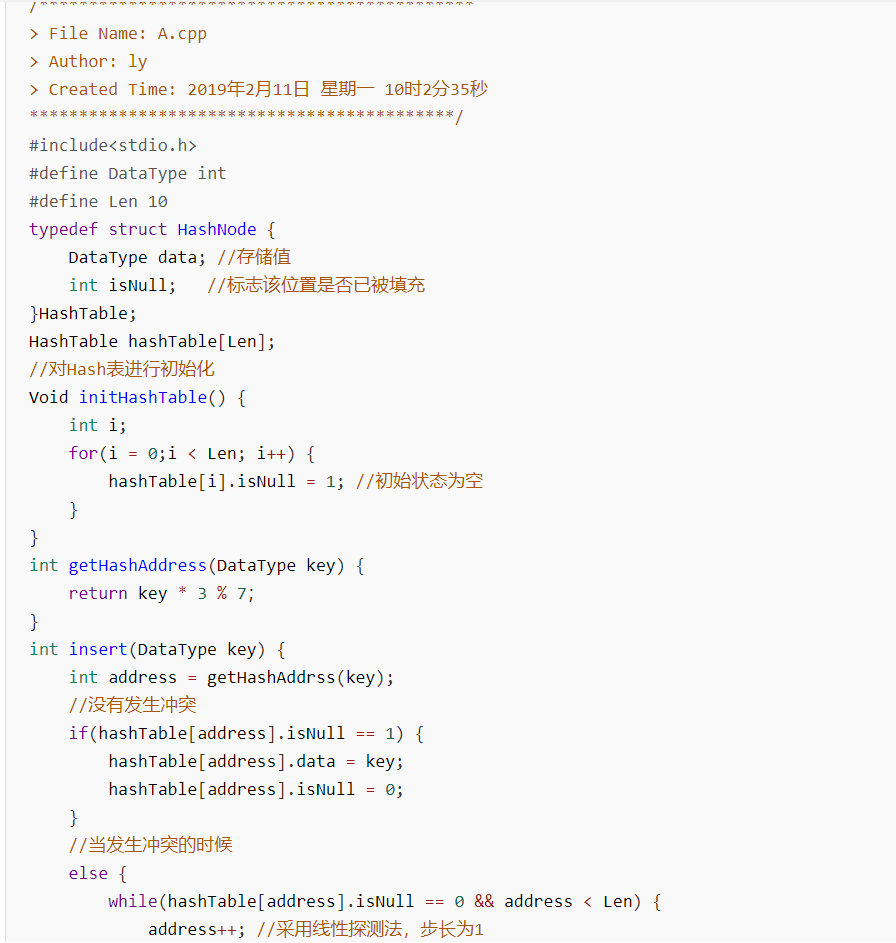
根据散列函数计算得到的散列地址可知，关键字27、13、55、32、18插入的地址均为开放地址，将它们直接插入到T[5],T[2],T[0],T[10],T[7]中。当插入关键字49时，散列地址5已被同义词27占用，因此探查h1=(5+1)% 11=6，此地址为开放地址，因此可将49插入到T[6]中；当插入关键字24时，其散列地址2已被同义词13占用，故探测地址h1=(2+1) % 11=3, 此地址为开放地址，因此可将24插入到T[3]中；当关键字38插入时，散列地址5已被同义词27占用，探查h1=(5+1) % 11=6,也被同义词49占用，再探查h2=(5+2)%11=7，地址7已被非同义词占用，因此需要再探查h3=(5+3) % 11=8，此地址为开放地址，因此可将38插入到T[8]中；当插入关键字43时，计算得到散列地址10已被关键字32占用，需要探查h1=(10+1)% 11=0，此地址已被占用，探查h2=(10+1) % 11=1为开放地址，因此可将43插入到T[1]中；

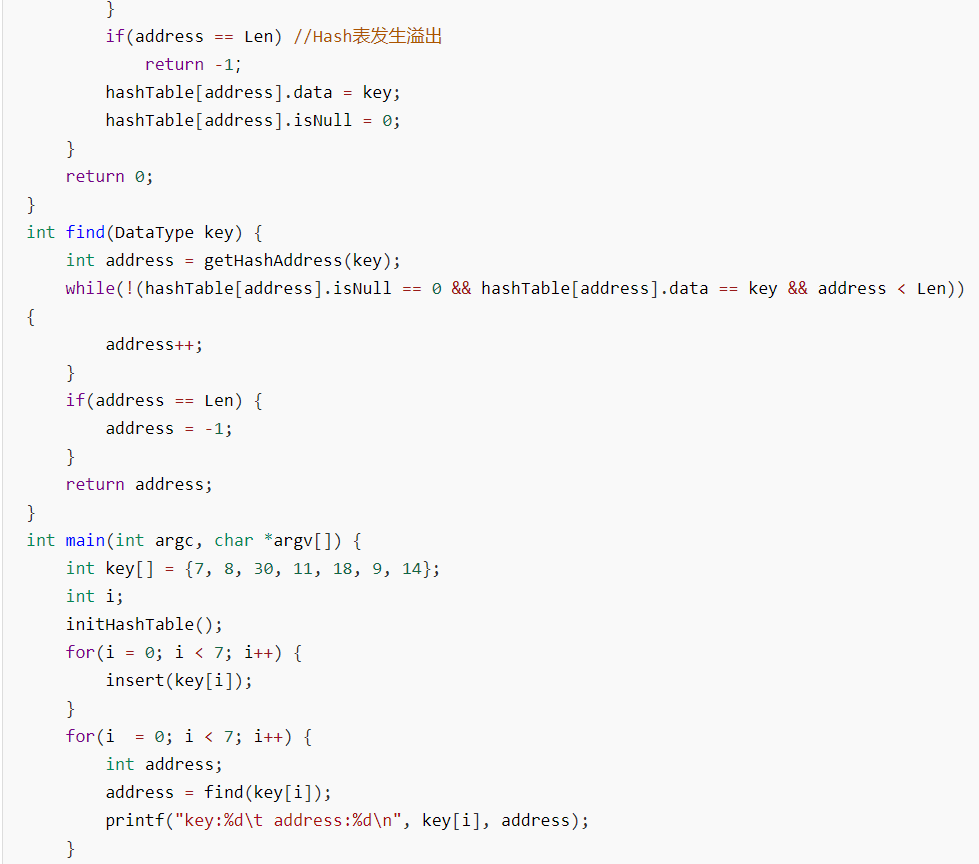
对用线性探测法构造的散列表的平均查找长度为：  
ASL=(1×5＋2×2＋3×14×1)/9 ≈1.78  
而用拉链法构造的散列表上查找成功的平均查找长度为：ASL=(1×5+2×3+3×1)/9≈1.55  
显然，开放定址法处理冲突的的平均查找长度要高于拉链法处理冲突的平均查找长度。但它们都比前面介绍的其它查找方法的平均查找长度要短。

编程题

1、将关键字序列{7, 8, 30, 11, 18, 9, 14}散列存储到散列表中。散列表的存储空间是一个下标从0开始的一维数组，长度为10，即{0, 1,2, 3, 4, 5, 6, 7, 8, 9}。散列函数为： H(key) = (key \* 3) % 7，处理冲突采用线性探测再散列法。

/\*采用数组实现哈希表\*/





2、You are given an integer array s[] and are asked to count how many positions a, b, c and d satisfy the condition: s[a] + s[b] + s[c] == s[d].

Note that a, b, c, and d do not need to be distinct.

输入

The first line of input contains an integer T, indicates the cases.

Each of the next T blocks contains an integer n first (0< n ≤1000 ), the length of the array s[], following with n integers representing s[] (0≤ si≤10^6 ).

输出

Output T lines each contains the answer required. You'd better use 'long long' instead of 'int'.

示例输入

4

4

0 0 0 0

2

1 3

4

1 10 100 111

1

3

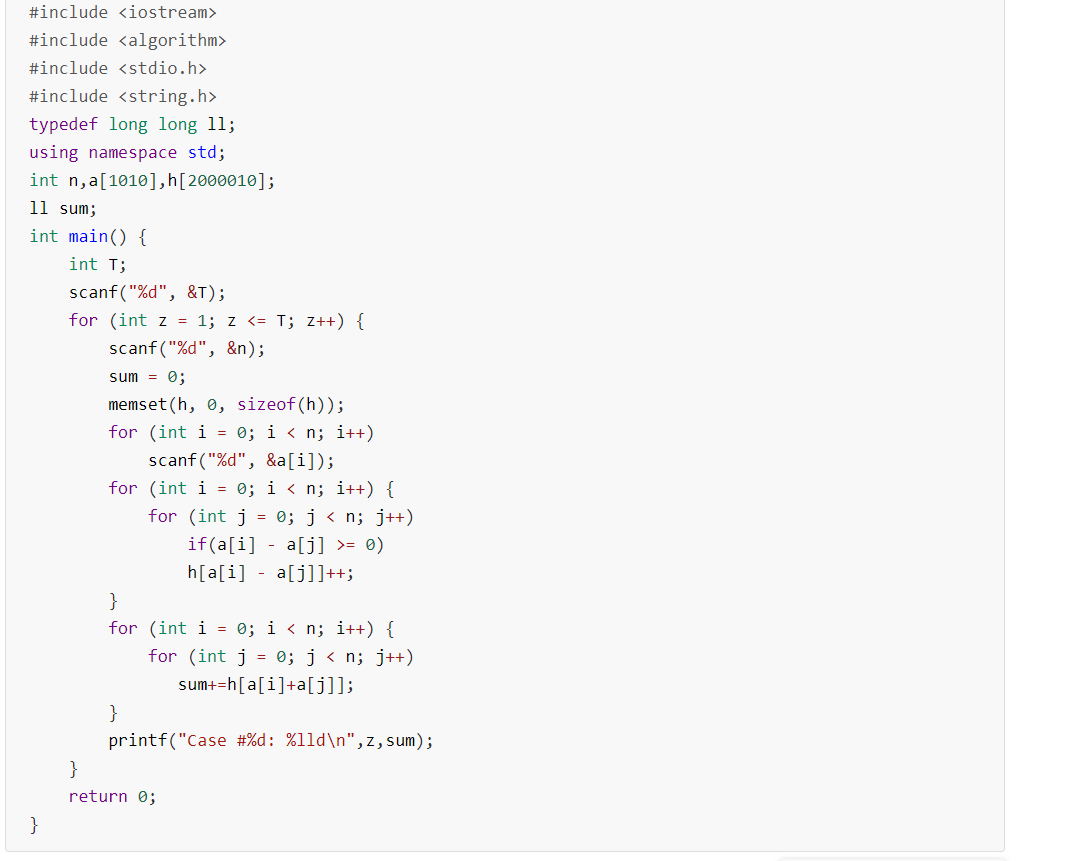
示例输出

Case #1: 256

Case #2: 1

Case #3: 6

Case #4: 0



3、给出n个正整数，然后有m个询问，每个询问一个整数，询问该整数是否在n个正整数中出现过。

输入描述 Input Description

第一行两个整数 n 和m。

第二行n个正整数（1<=n<= 100000）

第三行m个整数(1<=m<=100000)

输出描述 Output Description

一共m行，若出现则输出YES，否则输出NO

样例输入 Sample Input

4 2

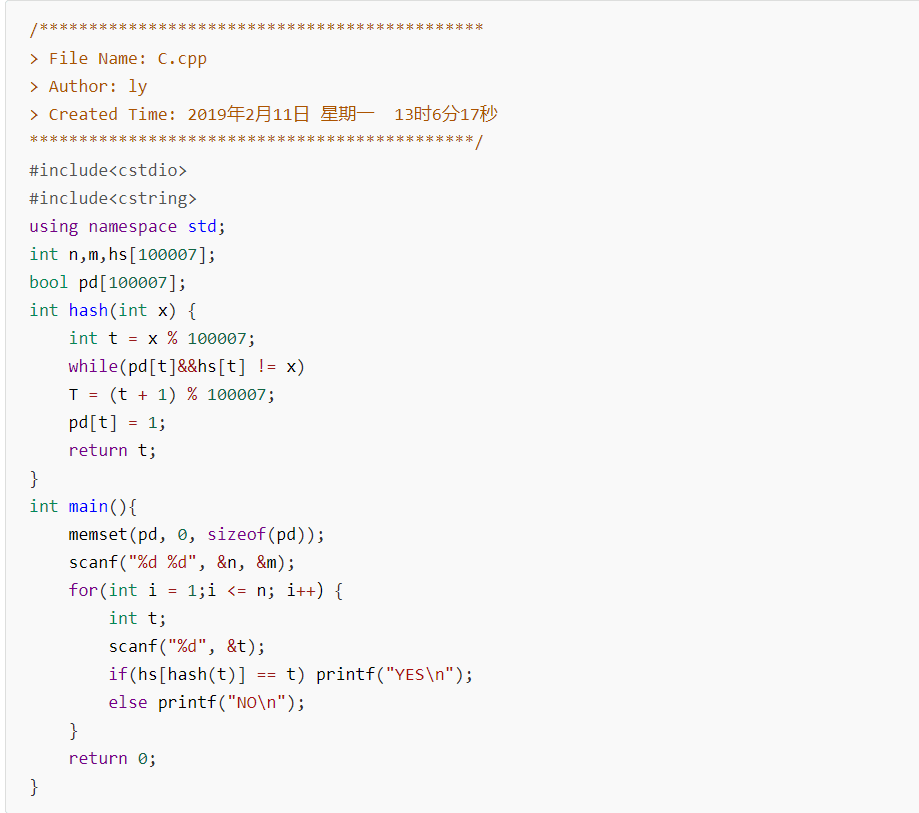
2 1 3 4

1 9

样例输出 Sample Output

YES

NO



4、codevs1553 互斥的数

题目描述 Description

有这样的一个集合，集合中的元素个数由给定的N决定，集合的元素为N个不同的正整数，一旦集合中的两个数x,y满足y = P\*x，那么就认为x,y这两个数是互斥的，现在想知道给定的一个集合的最大子集满足两两之间不互斥。

输入描述 Input Description

输入有多组数据，每组第一行给定两个数N和P（1<=N<=10^5, 1<=P<=10^9）。接下来一行包含N个不同正整数ai（1<=ai<=10^9）。

输出描述 Output Description

输出一行表示最大的满足要求的子集的元素个数。

样例输入 Sample Input

4 2

1 2 3 4

样例输出 Sample Output

3

---------------------



5、Problem Description

The French author Georges Perec (1936–1982) once wrote a book, La disparition, without the letter 'e'. He was a member of the Oulipo group. A quote from the book:

Tout avait Pair normal, mais tout s’affirmait faux. Tout avait Fair normal, d’abord, puis surgissait l’inhumain, l’affolant. Il aurait voulu savoir où s’articulait l’association qui l’unissait au roman : stir son tapis, assaillant à tout instant son imagination, l’intuition d’un tabou, la vision d’un mal obscur, d’un quoi vacant, d’un non-dit : la vision, l’avision d’un oubli commandant tout, où s’abolissait la raison : tout avait l’air normal mais…

Perec would probably have scored high (or rather, low) in the following contest. People are asked to write a perhaps even meaningful text on some subject with as few occurrences of a given “word” as possible. Our task is to provide the jury with a program that counts these occurrences, in order to obtain a ranking of the competitors. These competitors often write very long texts with nonsense meaning; a sequence of 500,000 consecutive 'T's is not unusual. And they never use spaces.

So we want to quickly find out how often a word, i.e., a given string, occurs in a text. More formally: given the alphabet {'A', 'B', 'C', …, 'Z'} and two finite strings over that alphabet, a word W and a text T, count the number of occurrences of W in T. All the consecutive characters of W must exactly match consecutive characters of T. Occurrences may overlap.

Input

The first line of the input file contains a single number: the number of test cases to follow. Each test case has the following format:

One line with the word W, a string over {'A', 'B', 'C', …, 'Z'}, with 1 ≤ |W| ≤ 10,000 (here |W| denotes the length of the string W).

One line with the text T, a string over {'A', 'B', 'C', …, 'Z'}, with |W| ≤ |T| ≤ 1,000,000.

Output

For every test case in the input file, the output should contain a single number, on a single line: the number of occurrences of the word W in the text T.

Sample Input

3

BAPC

BAPC

AZA

AZAZAZA

VERDI

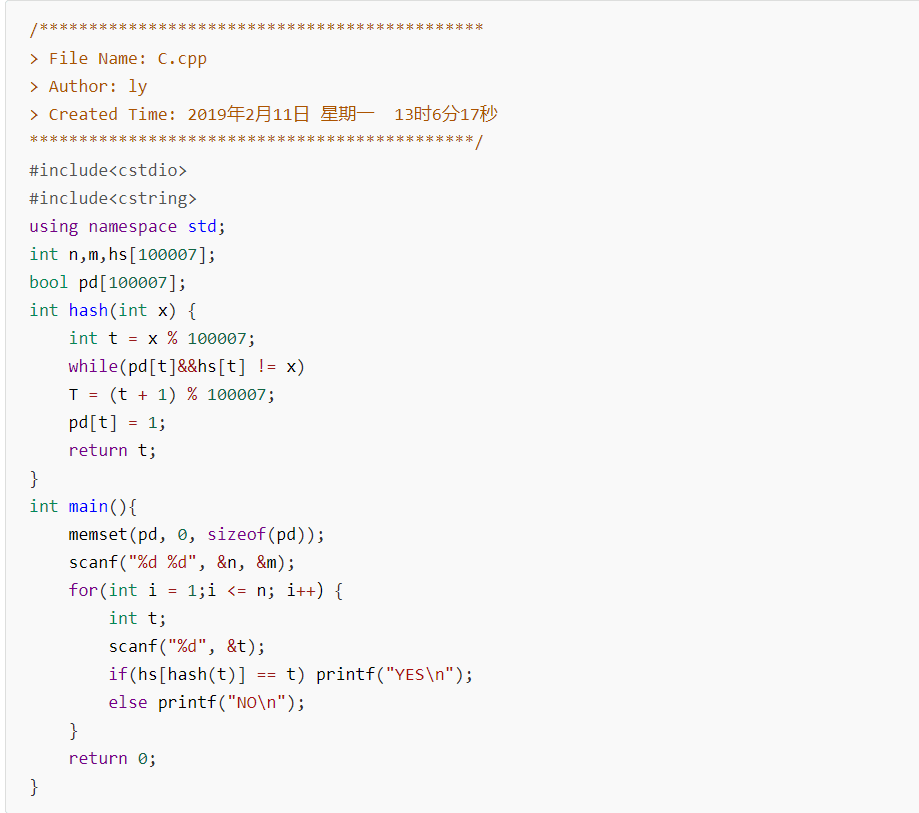
AVERDXIVYERDIAN

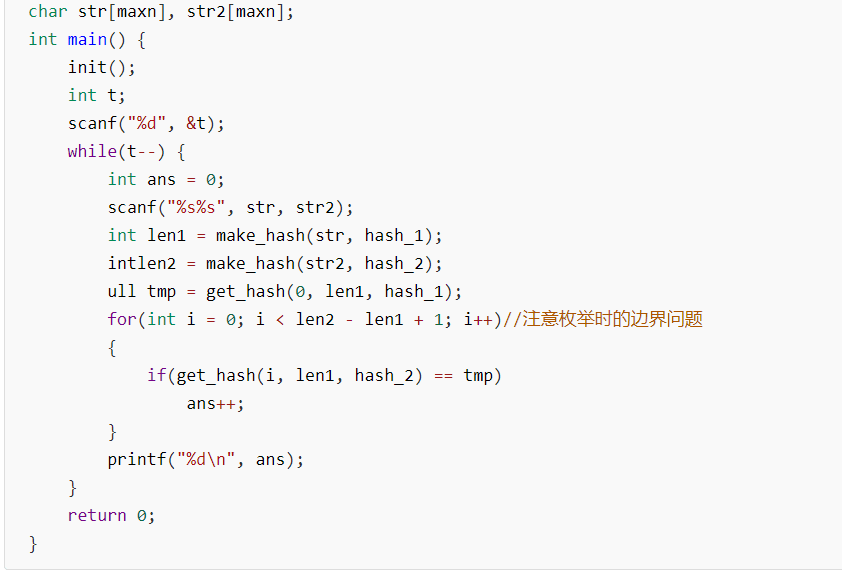
Sample Output

1

3

0





6、给定N个字符串（第i个字符串长度为Mi，字符串内包含数字、大小写字母，大小写敏感），请求出N个字符串中共有多少个不同的字符串。

输入格式：

第一行包含一个整数N，为字符串的个数。

接下来N行每行包含一个字符串，为所提供的字符串。

输出格式：

输出包含一行，包含一个整数，为不同的字符串个数。

输入输出样例

输入样例#1：

5

abc

aaaa

abc

abcc

12345

输出样例#1：

4

#include<bits/stdc++.h>

typedef long long ll;

typedef unsigned long long ull;

using namespace std;

const int maxn =1e4+7;

ull mod=212370440130137957ll;//质数

ull base=131;

int prime=233317;

ull a[maxn];

char s[maxn];

ull Hash(char str[]) {

int len = strlen(str);

ull ans = 0;

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

Ans = (ans \* base + (ull)str[i]) % mod + prime;

return ans;

}

int main() {

int N, ans = 1;

scanf("%d", &N);

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

scanf("%s", s);

a[i] = Hash(s);

}

sort(a + 1, a + 1 + N);

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

if(a[i] != a[i + 1])

ans++;

}

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

return 0;

}