Problem: Sherlock & Anagram

Given a string , find the number of "[unordered](http://en.wikipedia.org/wiki/Unordered_pair) [anagrammatic](http://en.wikipedia.org/wiki/Anagram) pairs" of substrings. In other words, find the number of *unordered* pairs of substrings of  that are anagrams of each other.

Two strings are **anagrams** of each other if the letters of one string can be rearranged to form the other string.

**Input Format**   
First line contains , the number of testcases. Each testcase consists of string  in one line.

**Constraints**   
   
   
String  contains only the lowercase letters of the English alphabet.

**Output Format**   
For each testcase, print the required answer in one line.

**Sample Input 0**

2

abba

abcd

**Sample Output 0**

4

0

**Sample Input 1**

5

ifailuhkqq

hucpoltgty

ovarjsnrbf

pvmupwjjjf

iwwhrlkpek

**Sample Output 1**

3

2

2

6

3

**Explanation**

**Sample 0**   
Let's say  denotes the substring .

testcase 1:   
For S=abba, anagrammatic pairs are:  (a and a),  (ab and ba),  (b and b) and  (abb and bba).

testcase 2:   
No anagrammatic pairs.

**Sample 1**   
Left as an exercise to you.

Solution

int combinations(int number)

{

return ( number \* (number-1) ) / 2;

}

int main() {

int cases;

cin>>cases;

//Feeding and processing the data

for(int a=0; a<cases; a++)

{ long count=0;

vector <string> store;

string str;

cin>>str;

int length = str.length();

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

{

for(int k=1; k<=length-j+1; k++)

{

string temp = str.substr(k-1, j);

sort(temp.begin(), temp.end());

store.push\_back(temp);

}

}

sort(store.begin(), store.end());

string temp;

int size = store.size();

for(int i=0; i<size-1; i++)

{

int combi = 1;

if(store[i]==store[i+1])

{

temp=store[i];

while(store[i+1]==temp)

{

combi+=1;

i+=1;

}

}

count+=combinations(combi);

}

cout<<count<<endl;

}

return 0;

}

`’Anshul AgGarwal