基础知识：

一. sort排序:

将数组中下标范围为[n1,n2)的元素从小到大排序。下标为n2的元素不在排序区间内

eg.int a[] = {15,4,3,9,7,2,6}; sort(a+2,a+5); //结果：{15,4,3,7,9,2,6}　　//默认从小到大排序

eg.int a[] = {15,4,3,9,7,2,6}; sort(a+1,a+4,greater()); // 结果：{15,9,4,3,7,2,6}

//大到小排序

eg.自定义排序:

```
struct  结构体名
{
        bool operator()(const T & a1,const T & a2){
            //若a1应该在a2前面，则返回true.
            //否则返回false.
        }
};

Student students [] = { {"Jack",112,3.4},{"Mary",102,3.8},{"Mary",117,3.9},
{"Ala",333,3.5},{"Zero",101,4.0}};
struct StudentRule3
{       //按gpa从高到低排
        bool operator() (const Student & s1,const Student & s2)
        {
            return s1.gpa > s2.gpa;
        }
};
```

二.在排好序的数组上进行二分查找的算法:binary_search,lower_bound,upper_bound

在从小到大排好序的基本类型数组上进行二分查找

binary_search(数组名+n1，数组名+n2,值);　n1和n2都是int类型的表达式，可以包含变量 如果n1=0,则 + n1可以不写

查找区间为下标范围为[n1,n2)的元素，下标为n2的元素不在查找区间内 在该区间内查找"等于"值”的元素，返回值为true(找到）或false(没找到）

"等于"的含义： a 等于 B <=> a < b和b < a都不成立。

在自定义排序规则时:

在用自定义排序规则排好序的、元素为任意的T类型的数组中进行二分查找

binary_search(数组名+n1，数组名+n2,值，排序规则结构名());

n1和n2都是int类型的表达式，可以包含变量 如果n1=0,则 + n1可以不写 查找区间为下标范围为[n1,n2)的元素，下标为n2的元素不在查找区间内

在该区间内查找"等于"值的元素，返回值为true(找到）或false(没找到）

查找时的排序规则，必须和排序时的规则一致!如果不一致，则找到的值没有意义，为0!

"等于"的含义： a 等于 b "a必须在b前面"和"b必须在a前面"都不成立

eg.

```
#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;
struct Rule //按个位数从小到大排
{
        bool operator()( const int & a1,const int & a2) {
                return a1%10 < a2%10;
        }
};
void Print(int a[],int size) {
        for(int i = 0;i < size;++i) {
                cout << a[i] << "," ;
        }
        cout << endl;
}
int main() {
        int a[] = { 12,45,3,98,21,7};
        sort(a,a+6);
        Print(a,6);
        cout <<"result:"<< binary_search(a,a+6,12) << endl;
        cout <<"result:"<< binary_search(a,a+6,77) << endl;
        sort(a,a+6,Rule()); //按个位数从小到大排
        Print(a,6);
        cout <<"result:"<< binary_search(a,a+6,7) << endl;
        cout <<"result:"<< binary_search(a,a+6,8,Rule()) << endl;
        return 0;
}
```

```
3,7,12,21,45,98,
result:1
result:0
21,12,3,45,7,98,
result:0
result:1
```

"等于"的含义： a 等于 b <=> "a必须在b前面"和"b必须在a前面"都不成立

eg2.

```
#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;
struct Student {
        char name[20];
        int id;
        double gpa;
};

Student students [] = {
        {"Jack",112,3.4},{"Mary",102,3.8},{"Mary",117,3.9},
        {"Ala",333,3.5},{"Zero",101,4.0}};
```

```
struct StudentRule1 { //按姓名从小到大排
        bool operator() (const Student & s1,const Student & s2) {
                if( stricmp(s1.name,s2.name) < 0)
                        return true;
                return false;
        }
};
struct StudentRule2 { //按id从小到大排
        bool operator() (const Student & s1,const Student & s2) {
                return s1.id < s2.id;
        }
};
struct StudentRule3 {//按gpa从高到低排
        bool operator() (const Student & s1,const Student & s2) {
                return s1.gpa > s2.gpa;
        }
};
int main(){
    Student s;
    strcpy(s.name,"Mary");
    s.id= 117;
    s.gpa = 0;
    int n = sizeof(students) / sizeof(Student);
    sort(students,students+n,StudentRule1()); //按姓名从小到大排
    cout << binary_search( students , students+n,s,
                                        StudentRule1()) << endl;
    strcpy(s.name,"Bob");
    cout << binary_search( students , students+n,s,
                                        StudentRule1()) << endl;
    sort(students,students+n,StudentRule2()); //按id从小到大排
    cout << binary_search( students , students+n,s,
                                        StudentRule2()) << endl;
    return 0;
}
```

101

lower_bound二分查找下界：

在对元素类型为T的从小到达排好序的基本类型的数组中进行查找:

T*lower_bound(数组名+n1,数组名+n2,值);

返回一个指针T *p;*p是查找区间里面下标最小的，大于等于“值”的元素。如果找不到，p指向下标为n2的元素

用法二：在元素为任意的T类型，按照自定义排序规则排好序的数组中进行查找

T*lower_bound(数组名+n1,数组名+n2,值，排序规则结构名());返回一个指针T*p;*p是查找区间里下标最小的，按自定义排序规则，可以排在“值”后面的元素。如果找不到，p指向下标为n2的元素。

upper_bound二分查找上界：

在对元素类型为T的从小到大排好序的基本类型的数组中进行查找

T*upper_bound(数组名+n1,数组名+n2,值);

返回一个指针T *p;*p是查找区间里面下标最小的，大于“值”的元素。如果找不到，p指向下标为n2的元素

用法二：在元素为任意的T类型，按照自定义排序规则排好序的数组中进行查找

T*upper_bound(数组名+n1,数组名+n2,值，排序规则结构名());返回一个指针T*p;*p是查找区间里下标最小的，按自定义排序规则，必须排在“值”后面的元素。如果找不到，p指向下标为n2的元素。

eg.

```
#include <iostream>
#include <cstring>
#include <algorithm>
using namespace std;
struct Rule
{
	bool operator()( const int & a1,const int & a2) {
		return a1%10 < a2%10;
	}
};
void Print(int a[],int size) {
	for(int i = 0;i < size;++i) {
		cout << a[i] << "," ;
	}
	cout << endl;
}

#define NUM 7
int main()
{
	int a[NUM] = { 12,5,3,5,98,21,7};
	sort(a,a+NUM);
	Print(a,NUM); // => 3,5,5,7,12,21,98,
	int * p = lower_bound(a,a+NUM,5);
	cout << *p << "," << p-a <<  endl; //=> 5,1
	p = upper_bound(a,a+NUM,5);
	cout << *p  << endl; //=>7
	cout << * upper_bound(a,a+NUM,13) << endl; //=>21
```

```
        sort(a,a+NUM,Rule());
        Print(a,NUM); //=>21,12,3,5,5,7,98,
        cout << * lower_bound(a,a+NUM,16,Rule()) << endl;  // => 7
        cout << lower_bound(a,a+NUM,25,Rule()) - a<< endl; // => 3
        cout << upper_bound(a,a+NUM,18,Rule()) - a << endl; // => 7
        if( upper_bound(a,a+NUM,18,Rule()) == a+NUM)
                cout << "not found" << endl;  //=> not found
        cout << * upper_bound(a,a+NUM,5,Rule())  << endl;  // =>7
        cout << * upper_bound(a,a+NUM,4,Rule())  << endl;  // =>5
        return 0;
}
```

STL中的平衡二叉树的数据结构：

1.有时候需要在大量增加、删除数据的同时，还要进行大量数据的查找。我们希望增加数据、删除数据、查找数据都能在log(n)复杂度完成。

2.排序+二分查找显然不行，因为加入新数据就要重新排序。因此，可以使用“平衡二叉树”数据结构存放数据，STL给出四种“排序容器”：

multiset,set,multimap,map。

multiset用法：

multiset<T> st; 定义了一个multiset变量st,st里面可以存放T类型的数据，并且能自动排序。开始st为空。排序规则：表达式“a<b"为true,则a排在b前面。

可用st.insert添加元素，st.find查找元素，st.erase删除元素，复杂度都是log(n).

eg.

```
#include <iostream>
#include <cstring>
#include <set> //使用multiset和set需要此头文件
using namespace std;
int main()
{
        multiset<int> st;
        int a[10]={1,14,12,13,7,13,21,19,8,8 };
        for(int i = 0;i < 10; ++i)
                st.insert(a[i]); //插入的是a [i]的复制品
        multiset<int>::iterator i; //迭代器，近似于指针
        for(i = st.begin(); i != st.end(); ++i)
                cout << * i << ",";
        cout << endl;
输出: 1,7,8,8,12,13,13,14,19,21,
```

```
        i = st.find(22);     //查找22，返回值是迭代器
        if( i == st.end())   //找不到则返回值为 end()
                cout << "not found" << endl;
        st.insert(22);  //插入 22
        i = st.find(22);
        if( i == st.end())
                cout << "not found" << endl;
        else
                cout << "found:" << *i <<endl;
        //找到则返回指向找到的元素的迭代器
```

```
输出:
not found
found:22
```

```
        i = st.lower_bound(13);
//返回最靠后的迭代器 it，使得[begin(),it)中的元素
//都在 13 前面 , 复杂度 log(n)
        cout << * i << endl;
        i = st.upper_bound(8);
//返回最靠前的迭代器 it，使得[it,end())中的元素
//都在 8 后面，复杂度 log(n)
        cout << * i << endl;
        st.erase(i); //删除迭代器 i 指向的元素，即12
        for(i = st.begin(); i != st.end(); ++i)
                cout << * i << ",";
        return 0;
}
```

1,7,8,8,12,13,13,14,19,21,

```
输出:
13
12
1,7,8,8,13,13,14,19,21,22,
```

multiset上的迭代器：multiset<T>::iterator p; p是迭代器，相当于指针，可用于指向multiset中的元素。访问multiset中的元素要通过迭代器。

与指针不同：multiset上的迭代器可++,--,用!=和==比较，不可比大小，不可加减整数，不可相减。

multiset<T> st;

st.begin()返回值类型为multiset<T>::iterator,是指向st中的头一个元素的迭代器，st.end()返回值类型为multiset<T>::iterator,是指向st中的最后一个元素后面的迭代器。对迭代器++,其就指向容器下一个元素，--则令其指向上一个元素。

自定义排序规则的multiset用法:

```
#include <iostream>
#include <cstring>
#include <set>
using namespace std;
struct Rule1 {
        bool operator()( const int & a,const int & b)     {
                return (a%10) < (b%10);
        }//返回值为true则说明a必须排在b前面
};
int main()  {
        multiset<int,greater<int> > st; //排序规则为从大到小
        int a[10]={1,14,12,13,7,13,21,19,8,8 };
        for(int i = 0;i < 10; ++i)
                st.insert(a[i]);
        multiset<int,greater<int> >::iterator i;
        for(i = st.begin(); i != st.end(); ++i)
                cout << * i << ",";
        cout << endl;
```

输出：21,19,14,13,13,12,8,8,7,1,

```
        multiset<int,Rule1 > st2;
        //st2的元素排序规则为：个位数小的排前面
        for(int i = 0;i < 10; ++i)
                st2.insert(a[i]);
        multiset<int,Rule1>::iterator p;
        for(p = st2.begin(); p != st2.end(); ++p)
                cout << * p << ",";
        cout << endl;
        p = st2.find(133);
        cout << * p << endl;

        return 0;
}
```

输出：

```
1,21,12,13,13,14,7,8,8,19,
13
```

```
        multiset<int,Rule1 > st2;
        //st2的元素排序规则为：个位数小的排前面
        for(int i = 0;i < 10; ++i)
                st2.insert(a[i]);
        multiset<int,Rule1>::iterator p;
        for(p = st2.begin(); p != st2.end(); ++p)
                cout << * p << ",";
        cout << endl;
        p = st2.find(133);
        cout << * p << endl;

        return 0;
}
```

输出：

```
1,21,12,13,13,14,7,8,8,19,
13
```

find(x)：在排序容器中找一个元素y，使得“x必须排在y前面”和“y必须排在x前面”都不成立

注意，这里根据自定义规则，返回的是个位数，133和13个位数相同，所以查找到了13.

eg2.

```
#include <iostream>
#include <cstring>
#include <algorithm>
#include <set>
using namespace std;
struct Student {
        char name[20];
        int id;
        int score;
};
Student students [] = { {"Jack",112,78},{"Mary",102,85},
        {"Ala",333,92},{"Zero",101,70},{"Cindy",102,78}};
struct Rule {
    bool operator() (const Student & s1,const Student & s2) {
        if( s1.score != s2.score)  return s1.score > s2.score;
        else   return (strcmp(s1.name,s2.name) < 0);
    }
};
```

44

```
int main()
{
        multiset<Student,Rule> st;
        for(int i = 0;i < 5;++i)
                st.insert(students[i]); //插入的是students[i]的复制品
        multiset<Student,Rule>::iterator p;
        for(p = st.begin(); p != st.end(); ++p)
           cout << p->score <<" "<<p->name<<" "
                 << p->id <<endl;
        Student s = { "Mary",1000,85};
        p = st.find(s);
        if( p!= st.end())
          cout << p->score <<" "<< p->name<<" "
               << p->id <<endl;
        return 0;
}
```

```
92 Ala 333
85 Mary 102
78 Cindy 102
78 Jack 112
70 Zero 101
85 Mary 102
```

45

同样的道理，之所以可以find(s)，并且找得到，是因为根据自定义规则，ID并不影响排序，两个Mary谁排前面后面都可以，符合等于的定义，所以可以查得到。

set:

set和multiset的区别在于容器里不能有重复元素，因为若a和b重复，a必须排在b前面和b必须排在a前面都不成立。

set插入元素可能不成功

```
#include <iostream>
#include <cstring>
#include <set>
using namespace std;
int main()
{
        set<int> st;
        int a[10] ={ 1,2,3,8,7,7,5,6,8,12 };
        for(int i = 0;i < 10; ++i)
                st.insert(a[i]);
        cout << st.size() << endl;  //输出: 8
        set<int>::iterator i;
        for(i = st.begin(); i != st.end(); ++i)
                cout << * i << ","; //输出: 1,2,3,5,6,7,8,12,
        cout << endl;
        pair<set<int>::iterator, bool> result =  st.insert(2);
        if( ! result.second ) //条件成立说明插入不成功
                cout << * result.first <<" already exists."
                     << endl;
        else
                cout << * result.first << " inserted." << endl;
        return 0;
}
```

输出:

```
2 already exists.
```

```
pair<set<int>::iterator, bool>

⇔

struct {
   set<int>::iterator first;
   bool second;
};
```

multimap:

multimap容器里面的元素，都是pair形式的。multimap<T1,T2> mp;则mp里的元素都是如下类型：

```
struct{
        T1 first; //关键字
        T2 second;//值
};
```

multimap中的元素按照first排序，并可以按first进行查找。缺省的排序规则是"a.first<b.first"为true,则a排在b前面

eg.

multimap的应用

一个学生成绩录入和查询系统，接受以下两种输入：

```
Add name id score
Query score
```

name是个不超过16字符的字符串，中间没有空格，代表学生姓名。id是个整数，代表学号。score是个整数，表示分数。学号不会重复，分数和姓名都可能重复。

两种输入交替出现。第一种输入表示要添加一个学生的信息，碰到这种输入，就记下学生的姓名、id和分数。第二种输入表示要查询，碰到这种输入，就输出已有记录中分数比score低的最高分获得者的姓名、学号和分数。如果有多个学生都满足条件，就输出学号最大的那个学生的信息。如果找不到满足条件的学生，则输出“Nobody”

输入样例：

```
Add Jack 12  78
Query 78
Query 81
Add Percy 9  81
Add Marry 8  81
Query 82
Add Tom 11 79
Query 80
Query 81
```

输出样例：

```
Nobody
Jack 12 78
Percy 9 81
Tom 11 79
Tom 11 79
```

```
#include <iostream>
#include <map> //使用multimap和map需要包含此头文件
#include <cstring>
using namespace std;
struct StudentInfo {
       int id;
       char name[20];
};
struct Student {
       int score;
       StudentInfo info;
};
typedef multimap<int,StudentInfo> MAP_STD;

// 此后 MAP_STD 等价于 multimap<int,StudentInfo>
// typedef int * PINT;
// 则此后 PINT 等价于 int *。 即 PINT p; 等价于  int * p;
int main()    {
      MAP_STD mp;
      Student st;
      char cmd[20];
      while( cin >> cmd ) {
            if( cmd[0] == 'A')  {
                cin >> st.info.name >> st.info.id >> st.score ;
                mp.insert(make_pair(st.score,st.info ));
            } //make_pair生成一个 pair<int,StudentInfo>变量
              //其first 等于 st.score, second 等于 st.info
            else if(  cmd[0] == 'Q' ){
                int score;
                cin >> score;
                MAP_STD::iterator p = mp.lower_bound (score);
                  if( p!= mp.begin()) {
                      --p;
                      score = p->first; //比要查询分数低的最高分
                      MAP_STD::iterator maxp = p;
                      int maxId = p->second.id;
                      for(; p != mp.begin() &&
                                p->first == score; --p) {
                          //遍历所有成绩和score相等的学生
                                if( p->second.id > maxId ) {
                                      maxp = p;
                                      maxId = p->second.id ;
                                }
                      }
```

```
                        if( p->first == score) {
//如果上面循环是因为 p == mp.begin() 而终止，则p指向的元素还要处理
                                if( p->second.id  > maxId ) {
                                        maxp = p;
                                        maxId = p->second.id ;
                                }
                        }
                        cout << maxp->second.name << " "
                             << maxp->second.id << " "
                             << maxp->first << endl;
                }
        //lower_bound的结果就是 begin，说明没人分数比查询分数低
                else cout << "Nobody" << endl;

        }
    }
    return 0;
}
```

map:

和multimap区别在于：不能有关键字重复的元素，可以使用[],下标为关键字，返回值为first和关键字相同的元素的second,并且插入元素可能失败

eg.

```
#include <iostream>
#include <map>
#include <string>
using namespace std;
struct Student {
        string name;
        int score;
};
Student students[5] = {
{"Jack",89},{"Tom",74},{"Cindy",87},{"Alysa",87},{"Micheal",98}};
typedef map<string,int> MP;
int main()
{
        MP mp;
        for(int i = 0;i < 5; ++i)
            mp.insert(make_pair(students[i].name,students[i].score));
        cout << mp["Jack"] << endl;  // 输出 89
        mp["Jack"] = 60; //修改名为"Jack"的元素的second
```

```
        for(MP::iterator i = mp.begin(); i != mp.end(); ++i)
                cout << "(" << i->first << "," << i->second << ") ";
//输出: (Alysa,87) (Cindy,87) (Jack,60) (Micheal,98) (Tom,74)
        cout << endl;
        Student st;
        st.name = "Jack";
        st.score =  99;
        pair<MP::iterator, bool> p =
                mp.insert(make_pair(st.name,st.score));
        if( p.second )
                cout << "(" << p.first->first << ","
                << p.first->second << ") inserted" <<endl;
        else
                cout << "insertion failed" << endl; //输出此信息
        mp["Harry"] = 78; //插入一元素，其first为"Harry",然后将其second改为78
        MP::iterator q = mp.find("Harry");
        cout << "(" << q->first << "," << q->second <<")" <<endl;
//输出 (Harry,78)
        return 0;
}
```

eg2.

map例题：单词词频统计程序

输入大量单词，每个单词，一行，不超过20字符，没有空格。 按出现次数从多到少输出这些单词及其出现次数。
出现次数相同的，字典序靠前的在前面

输入样例：

```
this
is
ok
this
plus
that
is
plus
plus
```

输出样例：

```
plus 3
is 2
this 2
ok 1
that 1
```

```
#include <iostream>
#include <set>
#include <map>
#include <string>
using namespace std;
struct Word {
	int times;
	string wd;
};
struct Rule {
	bool operator () ( const Word & w1,const Word & w2) {
		if( w1.times != w2.times)
			return w1.times > w2.times;
		else
			return w1.wd < w2.wd;
	}
};
int main()
{
	string s;
	set<Word,Rule> st;
	map<string,int> mp;
	while( cin >> s )
		++ mp[s] ;
	for( map<string,int>::iterator i = mp.begin();
	     i != mp.end(); ++i) {
		Word tmp;
		tmp.wd = i->first;
		tmp.times = i->second;
		st.insert(tmp);
	}
	for(set<Word,Rule>::iterator i = st.begin();
		i != st.end(); ++i)
	   cout << i->wd << " " << i->times << endl;
}
```

二.例题（来自于vj上chdacm挂的STL基础题）

A.where is the marble?

题目大意：输入两个数，n和m，n代表要输入多少个数，m 代表要查询几次，之后输入n个数。查询第几个数在哪个位置。一个很简单的题，做法很多，容器可选择也很多。

#include<iostream>

#include<algorithm>

```
using namespace std;
int main()
{
    int a[10010];
    int n,m;
    int x=1;
    while(cin>>n>>m)//输入
    {
        if(n==0)
            break;
        for(int i=0;i<n;i++)//输入n个数
        {
            cin>>a[i];
        }
        sort(a,a+n);//要求排序之后查位置
        int t;
        cout<<"CASE# "<<x++<<":"<<endl;
        for(int i=0;i<m;i++)
        {
            cin>>t;

    if(binary_search(a,a+n,t))//这里用不用binary_search都可以，因为是stl的题就用binary_search装了个逼
            {
                for(int j=0;j<n;j++)
                {
                    if(a[j]==t)
                    {
                        cout<<t<<" found at "<<j+1<<endl;

    break;//因为看题意是找到即开始下一个查找，题目中可能有多个相同的元素，所以记着break
                    }
                }
            }
            else//到最后都没有找到就找不到
            {
                cout<<t<<" not found"<<endl;
            }
        }
    }
    return 0;
}
```

Andy's First Dictionary

题目大意：给你一段话，有大写小写，统一转换成小写并按字典序排序，不能重复，输出。

```
#include<iostream>
#include<cstring>
#include<algorithm>
using namespace std;
void s_sort(char s[][300],int h)
{
    char t[300];
    for(int i=0;i<h;i++)//按字典序排序
    {
        for(int j=i+1;j<h;j++)
        {
            if(strcmp(s[j],s[i])<0)
            {
                strcpy(t,s[j]);
                strcpy(s[j],s[i]);
                strcpy(s[i],t);
            }
        }
    }
}
int main()
{
    char a[300],b[300],c[5010][300];
    int i,j,k,len,flag,h=0;
    while(gets(a))//无限输入文字，获得一行的内容
    {
        len=strlen(a);//获得这一行的长度
        for(i=0,j=0;i<=len;i++)
        {

    if((a[i]>='A'&&a[i]<='Z')||(a[i]>='a'&&a[i]<='z'))//因为要转换成小写，要对大小写判断转换，其实有一个tolower()函数挺方便的，不过下面的例题用到了
            {
                if(a[i]>='A'&&a[i]<='Z')
                {
                    b[j]=a[i]+32;//将转换成小写后的内容复制到b数组中
                    j++;
                }
                else
                {
                    b[j]=a[i];//是小写则直接复制
                    j++;
                }
            }
```

```
            else
            {
                if(a[i-1]>='A'&&a[i-1]<='Z'||a[i-1]>='a'&&a[i-1]<='z')//如果当前输入的内容不是大小写
                {
                    b[j]='\0';//则b数组结束
                    j=0;//结束之后b计数清零

    for(k=0,flag=0;k<h;k++)//k从0行开始往后，将b中的内容拷给c数组
                    {

    if(!strcmp(b,c[k]))//如果相等则为strcmp返回0，if中为1进入if，flag=1，不进入下面的if。如果不相等则进入下面的if进行拷贝
                        {
                            flag=1;
                        }
                    }
                    if(!flag)
                    {
                        strcpy(c[h],b);
                        h++;//h代表之前输入内容的总行数
                    }
                }
            }
        }
    }
    s_sort(c,h);
    for(i=0;i<h;i++)
    {
        printf("%s\n",c[i]);
    }
    return 0;
}
```

Ananagrams

题目大意：给你一堆单词，只要字母相同即为同一单词，要求按字典序输出不同单词，输入#则表示输入结束。

```
#include<iostream>
#include<vector>
#include<map>
#include<algorithm>
using namespace std;

map<string,int> cnt;
```

```
vector<string> words;
string s_sort(const string& s)//这个函数真的很棒，用了const，用了ans变量进行转换，保证最后输出时还是大写
{
    string ans=s;
    for(int i=0;i<ans.length();i++)
    {

    ans[i]=tolower(ans[i]);//tolower函数，将所有大写转换成小写，对非字母字符不作处理，这个代码实现也挺简单的吧？。
    }
    sort(ans.begin(),ans.end());//按字典序排序，这样可以确定单词是不是相同
    return ans;//排序完返回ans。
}

int main()
{
    string s;
    while(cin>>s)//输入字符串
    {
        if(s[0]=='#')//如果是#则停止输入
            break;
        words.push_back(s);//放入vector容器中
        string r =s_sort(s);

    if(!cnt.count(r))//如果r的次数为0（也就是第一次的情况）。如果r的次数不是0则直接走下面的语句，计数器+1。
        {
            cnt[r]=0;//将map的第二成员也就是计数器赋值为0.
        }
        cnt[r]++;//之后在加一。
    }
    vector<string> ans;//声明一个vector容器
    for(int i=0;i<words.size();i++)//遍历整个words里面放的词语
    {

    if(cnt[s_sort(words[i])]==1)//找到与map第一关键字相同的且第二关键字（也就是计数器）为1的，就放入ans容器。
        {
            ans.push_back(words[i]);
        }
    }
    sort(ans.begin(),ans.end());//将ans容器中的单词按字典序排序
    for(int i=0;i<ans.size();i++)
```

```
    {
        cout<<ans[i]<<endl;
    }
    return 0;
}
```

Team Queue
题目大意：一个队列，加入元素时，该元素先看有没有和它是一块的，如果有，则进入和它一块的元素的最后面，不然则成为队列的最后一个。请自行脑补买饭插队那个意思。先输入一个n,n表示有多少组是一块的元素（就当认识），然后输入n组数据，第一个数字是m，m代表这一组认识的元素有多少个。然后会输入包括DEQUEUE（出队）,ENQUEUE（入队）,STOP（结束输入）等指令。这个题有一个映射的问题

```
#include<iostream>
#include<stdio.h>
#include<queue>
#include<map>
using namespace std;

int main()
{
    int n,kase=1;
    while(cin>>n)
    {
        if(n==0)
            break;
        cout<<"Scenario #"<<kase++<<endl;
        map<int ,int> team;
        for(int i=0;i<n;i++)
        {
            int m,mem;
            cin>>m;
            while(m--)
            {
                cin>>mem;
                team[mem]=i;//map的第二关键字记录了这个元素属于哪个小队列
            }
        }

        queue<int> q,q2[1010];
        while(1)
        {
            int x;
            char ord[10];
            scanf("%s", ord);
```

```
            if(ord[0]=='S')
                break;
            else if(ord[0]=='D')
            {
                int t = q.front();//要出队先找到总队列的队头
                printf("%d\n",q2[t].front()); //根据映射，找到和它认识的所有元素的
那个队列的队头
                q2[t].pop();
                if(q2[t].empty())如果小队列为空了，即和它认识的元素没有了
                {
                    q.pop();//总队列就要把这个小队列清除
                }
            }
            else if(ord[0]=='E')//插入元素
            {
                cin>>x;
                int t=team[x];//找到和它相熟悉的元素的队列
                if(q2[t].empty())//如果这个小队列里面没有和它认识的元素
                {
                    q.push(t);//在总队列里面单独加一个小队列，把它放在最后面
                }
                q2[t].push(x);//不然把它加入到和它认识的元素的队列的最后一个

            }
        }
        cout<<endl;
    }
    return 0;
}
```