各大容器的特点：

1.可以用下标访问的容器有（既可以插入也可以赋值）：vector、deque、map；

特别要注意一下，vector 和 deque 如果没有预先指定大小，是不能用下标法插入元素的！

2. 序列式容器才可以在容器初始化的时候制定大小，关联式容器不行；

3.注意，关联容器的迭代器不支持 it+n 操作，仅支持 it++操作。

序列式容器：

一、vector

当需要使用数组的情况下，可以考虑使用 vector

1.特点：

(1) 一个动态分配的数组(当数组空间内存不足时，都会执行: 分配新空间-复制元素-释放原空间);

(2) 当删除元素时，不会释放限制的空间，所以向量容器的容量(capacity)大于向量容器的大小(size);

(3) 对于删除或插入操作，执行效率不高，越靠后插入或删除执行效率越高;

(4) 高效的随机访问的容器。

2.创建 vecotr 对象：

(1) vector<int> v1;

(2) vector<int> v2(10);

3.基本操作：

v.capacity(); //容器容量

v.size(); //容器大小

v.at(int idx); //用法和[]运算符相同

v.push_back(); //尾部插入

v.pop_back(); //尾部删除

v.front(); //获取头部元素

v.back(); //获取尾部元素

v.begin(); //头元素的迭代器

v.end(); //尾部元素的迭代器

v.insert(pos,elem); //pos 是 vector 的插入元素的位置

v.insert(pos, n, elem) //在位置 pos 上插入 n 个元素 elem

v.insert(pos, begin, end);

v.erase(pos); //移除 pos 位置上的元素，返回下一个数据的位置

v.erase(begin, end); //移除[begin, end)区间的数据，返回下一个元素的位置

reverse(pos1, pos2); //将 vector 中的 pos1~pos2 的元素逆序存储二分查找

```
#include<iostream>
#include<algorithm>
#include<vector>
#include<deque>
#include<map>
#include<cstring>
using namespace std;
int   main()
{
```

```
    vector<int> v(10);
    int num;
    vector<int>::iterator beg = v.begin();
    vector<int>::iterator end = v.end()
    vector<int>::iterator mid = v.begin() + (end - beg) / 2;
    for (int i = 0; i < 10; i++)
    {
        v[i] = i;
    }
    cin >> num;
    sort(v.begin(), v.end());
    while (*mid != num &&   beg <= end)
    {
        if (num < *mid)
        {
            end = mid;
        }
        else
        {
            beg = mid + 1;
        }
        mid = beg + (end - beg) / 2;
    }
    if (*mid == num)
    {
        cout << "Find" << endl;
    }
    else
    {
        cout << "Not Find" << endl;
    }
    return 0;
}
```

二、deque

1.特点：

(1) deque(double-ended queue 双端队列);

(2) 具有分段数组、索引数组，分段数组是存储数据的，索引数组是存储每段数组的首地址;

(3) 向两端插入元素效率较高！

(若向两端插入元素，如果两端的分段数组未满，既可插入;如果两端的分段数组已满，则创建新的分段函数，并把分段数组的首地址存储到 deque 容器中即可)。

中间插入元素效率较低！

2. 创建 deque 对象

(1) deque<int> d1;

(2) deque<int> d2(10);
3. 基本操作：
(1) 元素访问：
d[i];
d.at[i];
d.front();
d.back();
d.begin();
d.end();
(2) 添加元素：
d.push_back();
d.push_front();
d.insert(pos,elem); //pos 是 vector 的插入元素的位置
d.insert(pos, n, elem) //在位置 pos 上插入 n 个元素 elem
d.insert(pos, begin, end)
(3) 删除元素：
d.pop_back();
d.pop_front();
d.erase(pos); //移除 pos 位置上的元素，返回下一个数据的位置
d.erase(begin, end); //移除[begin, end)区间的数据，返回下一个元素的位置
三、list
1. 特点：
(1) 双向链表
2.创建对象：
list<int> L1；
list<int> L2(10);
3.基本操作：
(1) 元素访问：
lt.front();
lt.back();
lt.begin();
lt.end();
(2) 添加元素：
lt.push_back();
lt.push_front();
lt.insert(pos, elem);
lt.insert(pos, n , elem);
lt.insert(pos, begin, end);
lt.pop_back();
lt.pop_front();
lt.erase(begin, end);
lt.erase(elem);
(3)sort()函数、merge()函数、splice()函数：
sort()函数就是对 list 中的元素进行排序;

merge()函数的功能是：将两个容器合并，合并成功后会按从小到大的顺序排列;
比如：lt1.merge(lt2); lt1 容器中的元素全都合并到容器 lt2 中。
splice()函数的功能是：可以指定合并位置，但是不能自动排序！
这些函数用到的次数较少，要用时再加深印象！！！

关联式容器：

1. 特点：
(1) 关联式容器都是有序的，升序排列，自动排序;
(2) 实现的是一个平衡二叉树，每个元素都有一个父节点和两个子节点，
左子树的所有元素都比自己小，右子树的所有元素都比自己大;
四、set/multiset
1. 特点：
构造 set 集合的主要目的是为了快速检索,去重与排序
(1) set 存储的是一组无重复的元素，而 multiset 允许存储有重复的元素;
(2) 如果要修改某一个元素值，必须先删除原有的元素，再插入新的元素。
2.创建对象：
set<T> s;
set<T, op(比较结构体)> s;　　//op 为排序规则，默认规则是 less<T>(升序排列)，或者是 greater<T>(降序规则)。
函数对象：
```
class Sum
{
public:
    int operator()(int a, int b){return a+b;}
};
Sum sum;  //利用了()运算符的重载
```
3. 基本操作：
```
s.size();     //元素的数目
s.max_size(); //可容纳的最大元素的数量
s.empty();    //判断容器是否为空
s.find(elem);  //返回值是迭代器类型
s.count(elem); //elem 的个数，要么是 1，要么是 0，multiset 可以大于一
s.begin();
s.end()
s.rbegin();
s.rend();
s.insert(elem);
s.insert(pos, elem);
s.insert(begin, end)
s.erase(pos);
s.erase(begin,end);
s.erase(elem);
s.clear();//清除 a 中所有元素；
```
pair 类模板
1. 主要作用是将两个数据组成一个数据，用来表示一个二元组或一个元素对，

两个数据可以是同一个类型也可以是不同的类型。
当需要将两个元素组合在一起时，可以选择构造 pair 对象，
set 的 insert 返回值为一个 pair<set<int>::iterator,bool>。bool 标志着插入是否成功，而 iterator 代表插入的位置，若该键值已经在 set 中，则 iterator 表示已存在的该键值在 set 中的位置。
如：set<int> a;
a.insert(1);
a.insert(2);
a.insert(2);//重复的元素不会被插入；
注意一下：make_pair()函数内调用的仍然是 pair 构造函数
set 中的 erase()操作是不进行任何的错误检查的，比如定位器的是否合法等等，所以用的时候自己一定要注意。
创建 pair 对象：
pair<int, float> p1; //调用构造函数来创建 pair 对象
make_pair(1,1.2); //调用 make_pair()函数来创建 pair 对象
pair 对象的使用：
pair<int, float> p1(1, 1.2);
cout<< p1.first << endl;
cout<< p1.second << endl;
顺序遍历：
set<int> a;
set<int>::iterator it=a.begin();
for(;it!=a.end();it++)
  cout<<*it<<endl;
反序遍历：
set<int> a;
set<int>::reverse_iterator rit=a.rbegin();
for(;rit!=a.rend();rit++)
cout<<*rit<<endl;
find(key_value);//如果找到查找的键值，则返回该键值的迭代器位置，否则返回集合最后一个元素后一个位置的迭代器，即 end();
如：int b[]={1,2,3,4,5}; set<int> a(b,b+5);
set<int>::iterator it;
it=a.find(3);
if(it!=a.end()) cout<<*it<<endl;
else cout<<“该元素不存在”<<endl;
it=a.find(10);
if(it!=a.end()) cout<<*it<<endl;
else cout<<“该元素不存在”<<endl;
定义比较函数
set 容器在判定已有元素 a 和新插入元素 b 是否相等时，是这么做的：
（1）将 a 作为左操作数，b 作为右操作数，调用比较函数，并返回比较值；
（2）将 b 作为左操作数，a 作为右操作数，再调用一次比较函数，并返回比较值。
也就是说，假设有比较函数 f(x,y)，要对已有元素 a 和新插入元素 b 进行比较时，会先进行 f(a,b)操作，再进行 f(b,a)操作，然后返回两个 bool 值。

如果 1、2 两步的返回值都是 false，则认为 a、b 是相等的，则 b 不会被插入 set 容器中；
如果 1 返回 true 而 2 返回 false，那么认为 b 要排在 a 的后面，反之则 b 要排在 a 的前面；
如果 1、2 两步的返回值都是 true，则可能发生未知行为。
（1）自定义比较结构体；
首先，定义比较结构体

```
struct myComp
{
    bool operator() (const 类型 &a, const 类型 &b)//重载“()”操作符
      {
          ……
          return ……;
      }
};
```

然后，定义 set：

```
set<类型，myComp> s；
```

（2）重载“<”操作符
首先，在结构体中，重载“<”操作符，自定义排序规则

```
struct 结构体
{
     bool operator < (const 结构体类型 &a)
     {
         …
         return（…）;
     }
};
```

然后，定义 set

```
set<类型> s；
```

【第四届蓝桥杯预选赛】错误票据
题目描述：
某涉密单位下发了某种票据，并要在年终全部收回。
每张票据有唯一的 ID 号。全年所有票据的 ID 号是连续的，但 ID 的开始数码是随机选定的。
因为工作人员疏忽，在录入 ID 号的时候发生了一处错误，造成了某个 ID 断号，另外一个 ID 重号。
你的任务是通过编程，找出断号的 ID 和重号的 ID。
假设断号不可能发生在最大和最小号。
要求程序首先输入一个整数 N(N<100)表示后面数据行数。
接着读入 N 行数据。
每行数据长度不等，是用空格分开的若干个（不大于 100 个）正整数（不大于 100000）
每个整数代表一个 ID 号。
要求程序输出 1 行，含两个整数 m n，用空格分隔。
其中，m 表示断号 ID，n 表示重号 ID
输入：
要求程序首先输入一个整数 N(N<100)表示后面数据行数。
接着读入 N 行数据。

每行数据长度不等，是用空格分开的若干个（不大于 100 个）正整数（不大于 100000）
每个整数代表一个 ID 号。
输出：
要求程序输出 1 行，含两个整数 m n，用空格分隔。
其中，m 表示断号 ID，n 表示重号 ID
样例输入：
2
5 6 8 11 9
10 12 9
样例输出：
7 9
五、map/multimap
去重类问题
可以打乱重新排列的问题
有清晰的一对一关系的问题
1. 特点：
(1) map 为单重映射、multimap 为多重映射;
(2) 主要区别是 map 存储的是无重复键值的元素对，而 multimap 允许相同的键值重复出现，既一个键值可以对应多个值。
(3) map 内部自建了一颗红黑二叉树，可以对数据进行自动排序，所以 map 里的数据都是有序的，这也是我们通过 map 简化代码的原因。
(4)自动建立 key-value 的对应关系，key 和 value 可以是你需要的任何类型。
(5) key 和 value 一一对应的关系可以去重。
2. 创建对象：
map<T1,T2> m;
map<T1,T2, op> m; //op 为排序规则，默认规则是 less<T>
3. 基本操作：
m.at(key);
m[key];
m.count(key);
m.max_size(); //求算容器最大存储量
m.size(); //容器的大小
m.begin();
m.end();
m.insert(elem);
m.insert(pos, elem);
m.insert(begin, end);
注意一下：该容器存储的是键值对，所以插入函数与其他容器稍有不同
(1) 使用 pair<>构造键值对对象
map<int, float> m;
m.insert(pair<int, float>(10,2.3));
(2)使用 make_pair()函数构建键值对对象
map<int,float> m;

m.insert(make_pair(10,2.3));
(2) 使用 value_type 标志
map<int, float> m;
m.insert(map<int,float>::value_type(10,2.3));
m[key] = value; //m 只能是 map 容器，不适用于 multimap
m.erase(pos);
m.erase(begin,end);
m.erase(key);
使用 begin()和 end()遍历 map
使用数组的方法遍历 map
使用 find()查找
用 find 函数来定位数据出现位置它返回的一个迭代器。
当数据出现时，它返回数据所在位置的迭代器。
如果 map 中没有要查找的数据，它返回的迭代器等于 end 函数返回的迭代器。
map 在题目中的应用
去重：利用映射的一一对应性，把可能出现重复的数据设置为 key 值以达到去重的目的。
排序：自定义 Compare 类（依葫芦画瓢）
比如我建了一个学生-成绩的 map，原先是按照学生名字的字典序
排序的。
如果我想按照降序呢？学生姓名长度呢？
按照默认 cmp 的输出：
降序输出：
自定义 cmp 按照长度升序输出：
计数
假设定义一个 map<string,int>map1，输入数据 s，记为 first，如果这个数据存在，map1[s]++;
如果不存在，map1[s]=1;