关联容器：set 和 map

一、集合 set

集合 set 就是相同类型的元素构成的整体，分单重集(set)和多重集(multiset)两类：

◆单重集 set：每个元素至多出现一次。

◆多重集 multiset：每个元素可以重复出现。

使用 set 前，需要在程序头文件中包含声明“#include<set>”。

1、set 的定义

类似优先队列，有如下几种定义方式：

定义方法 1：缺省定义

#include<set> **set<int>s;** **multiset<int>ss;**	集合中的元素由小到大有序

定义方法 2：仿函数自定义小于符号

```
struct mycmp
{
  bool operator ()(int a, int b)
  {
       return  a > b;  //如果 a>b 则 a 排在前
  }
};
set< int, mycmp >s;
   元素类型  比较函数
```

定义方法 3、元素是结构体的优先队列定义

```
struct node
{
    int x, y;
    friend bool operator< (node a, node b) //通过重载<操作符来比较元素的大小>。
    {
         return a.x > b.x;   //结构体中，成员 x 值大排字在前面
    }
};
set<node>q;    //定义方法
```

也可以这样定义：

```
struct node{ int x, y; };  //结构体，作为 set 中的元素
struct mycmp   //自定义比较函数
{
    bool operator ()(node a, node b)
    {
        return  a.x > a.x;
    }
};
set<node,mycmp>s;    //定义方法
```

还有其他定义方法，不一一穷举

2、常用操作

操作	含义
s.size()	set 中的元素个数
set<int>::iterator it;	容器迭带器（指针）
s.begin()	容器中第一个元素的地址（迭带器）
s.end()	容器中最后一个元素后的地址迭带器）
s.insert(x)	向 set 容器中插入一个元素，成功返回 1，否则返回 0
s.erase(x)	删除 set 中值为 x 的元素，成功返回 1，否则返回 0
s.erase(it)	删除迭带器指向的元素，成功返回 1，否则返回 0
s.clear()	清空 set
it=s.find(x)	查找 x 在容器中出现的位置（迭带器），若 it==s.end()，则表示不存在
s.count(x)	计算 x 在容器中出现次数，0 表示不存在
it=s.lower_bound(x)	返回大于等于 x 的第一个元素的跌代器，返回 s.end()表示没找到
it=s.upper_bound(x)	返回大于 x 的第一个元素的跌代器，返回 s.end()表示没找到

set 中的插入、删除、查找的时间复杂度都是 $\log_2 n$ 级的。

二、映射 map

映射 map 就是从键到值得映射，他是数组的高级版本。使用 map 前，需要在程序头文件中包含声明“#include<map>”。

1、map 的定义

```
#include<map>
map<double,int>mp;
multimap<double,int>mmp;
```

```
//double 是第一关键字的类型(first)
//int 是关键字对应得值(second)
```

2、常用操作

操作	含义
mp.size()	map 中的元素个数
map<double,int>::iterator it;	容器迭带器（指针）
mp.begin()	容器中第一个元素的地址
mp.end()	容器中最后一个元素后的地址
mp[x]=y	其中 x 是 double ， y 是整数。例如：mp[0.1]=10 ；
mp.erase(x)	删除 mp 中第一关键字的值为 x 的元素，成功返回 1，否则返回 0
mp.erase(it)	删除 mp 中 it 指向的元素，成功返回 1，否则返回 0
mp.clear()	清空 map 容器
it=mp.find(x)	查找 x 在容器中出现的位置（迭带器），若 it==mp.end()，则表示不存在
mp.count()	计算 x 在容器中出现次数，0 表示不存在
it=mp.lower_bound(x)	返回大于等于 x 的第一个元素的跌代器，返回 mp.end()表示没找到
it=mp.upper_bound(x)	返回大于 x 的第一个元素的跌代器，返回 mp.end()表示没找到

map 中的插入、删除、查找的时间复杂度都是 $\log_2 n$ 级的。

例 1：集合维护[2]

请设计一种数据结构，完成如下的集合维护操作。注意：在操作中无论任何时候集合中元素互不相同。

1 x： 把整数 x 加入到集合中（如果已经存在元素 x，则不加入）；

2 x： 删除集合中的整数 x，如果不存在，则忽略次操作

3 x： 查询 x 是否属于集合，属于则输出"Yes"，否则输出"No"

4 x： 查询并输出集合中大于等于 x 的最小元素，如果不存在则输出"no"

5 x： 查询并输出集合中大于 x 的最小元素，如果不存在则输出"no"

6： 查询并输出集合中的最小元素值，若集合为空，则不输出。

7： 查询并输出集合中的最大元素值，若集合为空，则不输出。

8 x y：把集合中元素 x 修改成 y，如果集合中不存在 x，则忽略这个操作。

【输入格式】

第一行包含一个整数 m，表示有 m 个查询操作。再接下来的 m 行，每行表示一个操作。

1<=m<=500000，任何时候容器中整数元素的绝对值不会超过 2*10^9。

【输出格式】

按输入顺序输出操作 3、4、5、6、7 的结果。所有操作结束后，由小到大输出集合的所有元素的值。

【输入样例】

15 1 5 1 2 2 4 1 10 2 2 1 8 1 9 1 3 2 8 1 7 3 5 4 6 5 10 6 5 7	Yes 7 no 5 9 3 5 7 9 10

【讲解】

本题是 Set 和 map 的基本练习题：

Set 的代码

```
int main()
{
   scanf("%d",&m);
   set<int>S;
   set<int>::iterator p;
   for(int i=0;i<m;i++)
   {
      scanf("%d",&op);
      if(op==1)
      {
          scanf("%d",&x);
          S.insert(x); //因为是 set，自动去重复
      }
     else if(op==2)
     {
          scanf("%d",&x);
```

map 的代码

```
int main()
{
  scanf("%d",&m);
  map<int,int>mp;
  map<int,int>::iterator p;
  for(int i=0;i<m;i++)
  {
     scanf("%d",&op);
     if(op==1)
     {
          scanf("%d",&x);
          mp[x]=1; //因为是 map，自动去重复
     }
     else if(op==2)
     {
         scanf("%d",&x);
```

```
        S.erase(x);  //自动完成查找并删除
    }
    else if(op==3)
    {
        scanf("%d",&x);
        p=S.find(x);
        if(p!=S.end())
           printf("Yes\n");
        else
           printf("No\n");
    }
    else if(op==4)
    {
       scanf("%d",&x);
       p=S.lower_bound(x);
        if(p!=S.end())
          printf("%d\n",*p);
        else
          printf("No\n");
     }
    else if(op==5)
    {
       scanf("%d",&x);
       p=S.upper_bound(x);
       if(p!=S.end())
          printf("%d\n",*p);
      else
          printf("No\n");
    }
    else if(op==6)
    {
       if(S.size()>0)
         printf("%d\n",*S.begin());
    }
    else if(op==7)
    {
       if(S.size()>0)
      printf("%d\n",*(--S.end()));
    }
    else if(op==8)
    {
       scanf("%d %d",&x,&y);
       if(S.find(x)!=S.end())
        { S.erase(x); S.insert(y);}
    }
  }
  for(p=S.begin();p!=S.end();p++)
       printf("%d ",*p);
  return 0;
}
```

```
        mp.erase(x);
    }
    else if(op==3)
    {
        scanf("%d",&x);
        p=mp.find(x);
        if(p!=mp.end())
           printf("Yes\n");
        else
           printf("No\n");
    }
    else if(op==4)
    {
       scanf("%d",&x);
       p=mp.lower_bound(x);
        if(p!=mp.end())
           printf("%d\n",*p);
        else
           printf("No\n");
    }
    else if(op==5)
    {
       scanf("%d",&x);
       p=mp.upper_bound(x);
        if(p!=mp.end())
          printf("%d\n",p->first);
        else
          printf("No\n");
    }
    else if(op==6)
    {
       if(mp.size()>0)
      printf("%d\n",mp.begin()->first);
    }
    else if(op==7)
    {
       if(mp.size()>0)
       {p=--mp.end();
         printf("%d\n",p->first);}
       }
     else if(op==8)
     {
       scanf("%d %d",&x,&y);
       if(mp.find(x)!=mp.end())
        { mp.erase(x); mp[y]=1;}
     }
   }
   for(p=mp.begin();p!=mp.end();p++)
      printf("%d ",p->first);
}
```

例 2、序列的离散化（P2734）

对于一个数据元素很大的序列：A[1],A[2],…,A[n]，我们在处理这个序列时，往往只会关心序列中元素的大小关系。这时候为了处理方便，可把序列映射为 1..n 范围内的数，其中最小的元素映射为 1，第 2 小的数映射为 2,……。例如：

原 序 列：19827345 2000000053 73925 98203456 73925 19827345

映射序列：　2　　4　　1　　3　　1　　2

由此可以看出，映射序列于原序列元素间的大小关系是等价的，在这里我们把“映射序列”又称为“名次序列”。 所以，离散序列实质就是原序列每个元素用它的名次代替（相同的元素名次相同），离散化就是要得到序列的名次数组。

【输入格式】

第 1 行，为整数 n，表示序列有 n 个实数。第 2 行包含 n 个实数，表示序列 A[1]..A[n]。

1<=n<=100000。A[i]在 double 范围内。

【输出格式】

包含 n 个整数，与原序列等价的名次序列。

【输入样例】

6 10.5 8.1 9.22 3.33 6.23 8.1	5 3 4 1 2 3

【讲解】

离散化，把无限空间中无限的个体映射到有限的空间中去，以此提高算法的时空效率。

本题需要得到每个实数映射成一个 1..n 之间的整数，那么 map 容器正好能实现这个功能！

```
int n;
double A[100005];
int main()
{
      map<double,int>mp;
       map<double,int>::iterator it;
      scanf("%d",&n);
      for(int i=1;i<=n;i++)
      {
          scanf("%lf",&A[i]);
          mp[A[i]]=1;   //先映射成 1
      }
      int rank=1;
      for(it=mp.begin();it!=mp.end();++it)
        it->second=rank++; //依次映射
      for(int i=1;i<=n;i++)
         printf("%d ",mp[A[i]]);
    return 0;
}
```

另一中常用的离散化方法，效率要高些！

```
double B[100005];


memcpy(B,A,sizeof(A));
sort(B+1,B+1+n);
int cnt=unique(B+1,B+1+n)-B;
for(int i=1;i<=n;i++)
  rank[i]=lower_bound(B+1,B+cnt,A[i])-B;
```

去重：int cnt=unique(B+1,B+1+n)-B;

等价写法：

```
int cnt=2;
for(int i=2;i<=n;i++)
  if(B[i]!=B[i-1]) B[cnt++]=B[i];
```

例 3、冷血格斗场（noi.openjudge 3.9 数据结构之 C++STL 3344）

为了迎接 08 年的奥运会，让大家更加了解各种格斗运动，facer 新开了一家冷血格斗场。格斗场实行会员制，但是新来的会员不需要交入会费，而只要同一名老会员打一场表演赛，证明自己的实力。

我们假设格斗的实力可以用一个正整数表示，成为实力值，两人的实力值可以相同。另外，每个人都有一个唯一的 id，也是一个正整数。为了使得比赛更好看，每一个新队员都会选择与他实力最为接近的人比赛，即比赛双方的实力值之差的绝对值越小越好，如果有多个人的实力值与他差别相同，则他会选择 id 最小的那个。

不幸的是，Facer 一不小心把比赛记录弄丢了，但是他还保留着会员的注册记录。现在请你帮 facer 恢复比赛纪录，按照时间顺序依次输出每场比赛双方的 id。

【输入】

第一行一个数 n(0 < n <=100000)，表示格斗场新来的会员数（不包括 facer）。以后 n 行每一行两个数，按照入会的时间给出会员的 id 和实力值。一开始，facer 就算是会员，id 为 1，实力值 1000000000。所有会员的实力值都是 2*10^9 以内的正整数。

【输出】

N 行，每行两个数，为每场比赛双方的 id，新手的 id 写在前面。

【讲解】

每读入一个新会员要做两件事：

1、查找已有会员中，与新会员实力最接近的（用 lower_bound 查找后，紧接着的两个谁最接近）

2、把新会员加入已有会员中（如果不存在或已经存在但 id 大，则需要加入）

map 实现：

```
int main()
{
  int n;
  map<int,int>mp;
  map<int,int>::iterator it;
  scanf("%d",&n);
  mp[1000000000]=1;  //facer
  int id,pw,idx,t;
  for(int i=0;i<n;i++)
  {
     scanf("%d%d",&id,&pw);
     it=mp.lower_bound(pw);
     if(it==mp.end()) it--;

     t=abs(it->first - pw);  // 大于等于 pw 的最小元素值：it->first
     idx=it->second;
     if(it!=mp.begin())
     {
       it--;  //小于 pw 的最大元素值 it->first
       if(t > abs(it->first-pw) || (t==abs(it->first-pw)) && it->second<idx)
             idx=it->second;
    }
    printf("%d %d\n",id,idx);

     it=mp.find(pw);
     if(it==mp.end() || it->second>id)  //用 id 小的替换
         mp[pw]=id;
  }
  return 0;
}
```

Set 代码实现：

```
struct data    //set 的元素类型
{
    int p,id;
    friend bool operator<(data a,data b)
    {
        return a.p<b.p;
    }
};

int main()
{
      int n;
      set<data>s;
      set<data>::iterator it;
      scanf("%d",&n);
      s.insert((data){1000000000,1}); //facer
      int idx,t;
      data a;
      for(int i=0;i<n;i++)
      {
          scanf("%d%d",&a.id,&a.p);
          it=s.lower_bound(a);
          if(it==s.end()) it--;
          t=abs(it->p - a.p);
          idx=it->id;
          if(it!=s.begin())
          {
              it--;
              if(t>abs(it->p-a.p) || (t==abs(it->p-a.p)) && it->id<idx)
                  idx=it->id;
          }
          printf("%d %d\n",a.id,idx);
          it=s.find(a);
          if(it==s.end())
            s.insert(a);
          else if(it->id > a.id)  //替换
          {
             s.erase(it);
             s.insert(a);
          }
      }
    return 0;
}
```

算法的时间复杂度分析：无论是 set 还是 map 每次操作的时间复杂度都是$O(log_2 n)$，要进行 n 次操作，所以算法的时间复杂度是：$O(nlog_2 n)$。

请自行写出使用 multiset 和 multimap 的程序！

注意：本题的集合是动态变化的（不断有元素加入或替换），如果使用静态数组，时间复杂度将会很高！

1)、若使用顺序查找，加入元素的时间复杂度是 O(1)，但查找是 O(n)的。

2)、若排序使用二分查找，查找复杂度是 O(logn)的，但加入元素的时间复杂度是 O(n)的。

例 4、求和集（P2114）

给定一个整数集合 S，找出一个最大的 d，使得 a+b+c=d，其中 a，b，c，d 是 S 中的不同元素。

【输入格式】

输入包含多组数据。每组数据的第一行为集合内的元素个数 n（1≤n≤1 000），以下 n 行每行一个 -53 6870 912~+536 870 911 的整数。输入结束标志为 n=0。

【输出格式】

对于每组数据，输出最大的 d。如果无解，输出“no solutio”。

【数据范围】

1≤n≤1 000

【讲解】

一种经典算法思想解答这个问题：**中途相遇法**

因为：a+b+c=d ➔ a+b = d-c

那么中途相遇法的思路就是：把所有 a+b 的值做成一个集合（multiset），然后枚举 d 和 c，查找 d-c 是否在集合中存在。

```
int n,a[maxn];
int main()
{
  while(1)
  {
     scanf("%d",&n);
     if(n==0) break;
     for(int i=1;i<=n;i++)
       scanf("%d",&a[i]);

     multiset<data>s;
     multiset<data>::iterator p,q,it;
     for(int i=1;i<=n;i++)  //生成集合
     for(int j=i+1;j<=n;j++)
       s.insert((data){a[i]+a[j],i,j});

     int ans=-inf;
     for(int i=1;i<=n;i++)   //d=a[i]
     for(int j=1;j<=n;j++) if(i!=j)  //c=a[j]
     {
         data t=(data){a[i]-a[j],i,j};
         p=s.lower_bound(t);
         for(it=p;it!=s.end()&&it->v==t.v;++it)
           if(t.i!=it->i && t.i!=it->j)
           if(t.j!=it->i && t.j!=it->j)
           {
               ans=max(ans,a[i]);
               break;
           }
     }
     if(ans>-inf)
         printf("%d\n",ans);
      else
         printf("no solutio\n");
    }
    return 0;
}
```

```
struct data
{
   int v,i,j;
   bool operator<(data b)
   {
       return v<b.v;
   }
};
```

本题还可用静态数组实现集合排序+查找来实现，效率要高些！代码如下：

```
int tot=0;
for(int i=1;i<=n;i++)  //生成集合 S 并排序
for(int j=i+1;j<=n;j++)
    S[tot++]=(data){a[i]+a[j],i,j};
sort(S,S+tot);

int ans=-inf;
for(int i=1;i<=n;i++)   //d=a[i]
for(int j=1;j<=n;j++) if(i!=j)  //c=a[j]
{
   data t=(data){a[i]-a[j],i,j};
   int p=lower_bound(S,S+tot,t)-S;  //二分查找
   for(int k=p;k!=tot && S[k].v==t.v;++k)
      if(t.i!=S[k].i && t.i!=S[k].j)
     if(t.j!=S[k].i && t.j!=S[k].j)
     {
         ans=max(ans,a[i]);
         break;
     }
}
```

```
struct data
{
   int v,i,j;
   bool operator<(data b)
   {
       return v<b.v;
   }
}S[1000*1000+5];
```

例 5、唯一的雪花（P2788）

输入一个长度为 n(n<=10^6)的序列 A，找一个尽量长的连续子序列 A[L]..A[R]，使得序列中没有相同的元素。

【输入格式】

第一行一个整数 n，表示序列 A 长度，接下来的 N 行，按顺序给出序列 A[1]..A[n]，每行一个整数数，表示序列的元素。

【输出格式】

没有重复元素的连续序列最长长度。

【数据范围】

1<=n<=10^6 ， 0<=A[i]<=10^9

【讲解】

把连续子序列看成一个**窗口**，任何时候，窗口中的元素都具有唯一性，那么**滑动窗口**的维护如下：

```
map<int,int>mp;
map<int,int>::iterator it;
int ans=1,L=1,R;
for(R=1;R<=n;R++)
{
   while(mp.find(A[R]) !=mp.end())   //窗口左端点右移动
   {
       mp.erase(A[L]);
       L++;
    }
   mp[A[R]]=1;   //A[R]进入窗口
   ans=max(ans,R-L+1);
}
```

请自行用 set 实现上面的程序！

还可以用标记数组（简单的 hash 表）来实现：标记数组 vis[x]=1 表示元素属于窗口，否则不属于！但本题序列的元素值达到 10^9，标记数组开这么大不现实，但我们可以把序列离散化，把每个元素映射成对应的名次(1..n)，这样保持元素间的大小关系不变，然后就可以用标记数组了。

```
bool vis[1000005]

memcpy(B,A,sizeof(A));     //离散化
sort(B+1,B+1+n);
int cnt=unique(B+1,B+1+n)-B;
for(int i=1;i<=n;i++)
   A[i]=lower_bound(B+1,B+cnt,A[i])-B;

int ans=1,L=1,R;
for(R=1;R<=n;R++)
{
    while(vis[A[R]]>1) { vis[A[L]]--; L++; }
    vis[A[R]]++;
     ans=max(ans,R-L+1);
}
```

例 6、游客运输

风景迷人的阆中市，拥有 n 个美丽景点，编号为 1..n。由于慕名而来的游客越来越多，市政府特意安排了一辆容量为 C 的观光公交车，为有课提供便捷的交通服务。观光公交每天早上从 1 号景点出发，一次前往 2,3,…n 号景点，每天只开一趟。

设某天有 m 为游客，每位乘车 1 次从一个景点到达另一个景点，第 i 位游客希望从景点 Li 到达景点 Ri(1<=Li<=Ri<=n)。现在给你观光公交的容量 C，同事提供给你每位乘客的信息，请你计算这一天中最多能满足都少个人的愿望。

【输入】

第一行包含两个整数 n,m,C，分别表示景点数目、游客数目和观光公交的容量。

第 2 行到第 m+1 行，每行包含两个整数：Li,Ri，其中第 i+1 行表示奶牛 i 想从景点 Li 到达 Ri。

【输出】

一个整数，表示能满足的游客数量。

【样例】

10 8 3 8 10 1 9 2 10 6 8 5 8 6 7 3 5 4 6	6

【数据范围】

n,m,C<=200000

【讲解】

替换型的贪心，贪心分析如下：

把游客按出发点（线段的左端点）从左到右排序。然后一次考察每个游客，对于游客 i：

Step1、把在车上但应在 L[i]之前下车的乘客都下车（右端点<=L[i]的）。*//按 L[i]排序的妙处*

Step2、现在考虑 i 是否上车：

情况 1、当前车上的人数 tot<C，则游客 i 上车 tot++;

情况 2、当前车上人满，设 j 是当前车上最晚下车的乘客，如果 R[j]>R[i]（j 比 i 晚下车），则用 i 去替换这个游客。*//因为替换后，i 可以提前腾出位置，所以划算些！*

代码实现：

暴力查找体现在下面两个地方：

顺序查找车上 R[k]<=L[i]的乘客

顺序车上 R[k]最大的乘客

时间复杂度为：$O(m^2)$

优化：设置 multiset 来存储车上乘客信息：以乘客的右端点 R[i]为关键字，集合的首元素是 R[i]最小的乘客，集合的尾元素是 R[i]最大的乘客，于是：

查找 R[k]<=L[i]，直接查看集合首元素

查找 R[k]最大的乘客，直接查看集合尾元素。

时间复杂度为：$O(m \times \log_2 C)$

设置结构体 p[i].l、p[i].r 表示乘客信息

```
int n,m,C;
bool vis[maxn]; //标记车上乘客
void solve30()
{
   sort(p+1,p+1+m);  //按 p[i].l 排序
   int ans=0,tot=0;
   memset(vis,0,sizeof(vis));
   for(int i=1;i<=m;i++)
   {
      if(p[i].l==p[i].r)//特殊乘客
         {ans++; continue;}

      for(int k=1;k<i;k++)  //下车乘客
        if(vis[k] && p[k].r<=p[i].l)
          vis[k]=0, tot--;

      if(tot<C)  //车有空位
      {
          vis[i]=1,ret++,ans++;
      }
      else //车已坐满
      {
        int maxv=-1,j=-1;  //寻找替换对象
        for(int k=1;k<i;k++)
          if(vis[k] && p[k].r>maxv)
              maxv=p[k].r,j=k;

        if(maxv>p[i].r)
          vis[j]=0,vis[i]=1;
      }
   }
   printf("%d\n",ans);
}
```

```
void solve100()
{
    sort(p+1,p+1+m);
    multiset<int>S;
    multiset<int>::iterator it;
    int ans=0;
    for(int i=1;i<=m;i++)
    {
       if(p[i].l==p[i].r)
         { ans++; continue; }
       it=S.begin();
       while(!S.empty() && *it<=p[i].l)
       {
             S.erase(S.begin());
             it=S.begin();
       }

       if(S.size()<C)
       {
          ans++;
           S.insert(p[i].r);
       }
       else
       {
          it=--S.end();
          if(*it>p[i].r)
          {
              S.erase(it);
              S.insert(p[i].r);
          }
       }
    }
    printf("%d\n",ans);
}
```