STL序列式容器研讨

STL相关简述

STL:广义上定义为容器，算法，迭代器;

STL六大组件:容器，迭代器，算法，仿函数，适配器（配接器），空间配置器

容器：各种数据结构，如vector、list、deque等等

算法：各种常用算法，如sort、find、copy等等

迭代器：扮演容器与算法之间的胶合剂

仿函数：行为类似函数，可作为算法某种策略

适配器：一种用来修饰容器或者仿函数或迭代器接口的东西

空间配置器：负责空间的配置与管理

目录

序列式容器

向量(vector)

vector概述：Vector是一个封装了动态大小数组的顺序容器。跟任意其它类型容器一样，它能够存放各种类型的对象。可以简单的认为，向量是一个能够存放任意类型的动态数组。

vector特点：1.顺序序列。顺序容器中的元素按照严格的线性顺序排序。可以通过元素在序列中的位置访问对应的元素。

2.动态数组。支持对序列中的任意元素进行快速直接访问，甚至可以通过指针算述进行该操作。操供了在序列末尾相对快速地添加/删除元素的操作。

3.能够感知内存分配器的（Allocator-aware）。容器使用一个内存分配器对象来动态地处理它的存储需求。

vector数据结构：采用的是连续的线性空间，属于线性存储，采用三个迭代器来指向分配的线性空间的不同范围。

vector基本操作（特色）：

1.删除容器中的元素

```
vector<int> v1;//默认构造 无参构造
for(int i=0;i<10;i++){v1.push_back(i);}//尾插法插入元素
for(vector<int>::iterator it=v1.begin(); it!=v1.end(); ){if(*it == 7)
{it = v1.erase(it);}//删除指定位置的元素，注意不能频繁删除
else {++it;}}
v1.pop_back();//删除容器中最后一个元素
v1.clear();//清空容器中的元素，但不释放空间
```

2.容器大小与容器长度

vector<int> v1;

for(int i=0;i<10;i++){v1.push_back(i);}

v1.resize(int num);//重新指定容器的长度为num，若容器变长，则以默认值(0)填充新位置；若容器变短，则末尾超出容器长度的元素会被删除

v1.resize(int num,elem);//重新指定容器的长度为num，若容器变长，则以elem填充新位置；若容器变短，则末尾超出容器长度的元素会被删除

v1.reserve(int len);//容量预留len个元素长度，预留位置不初始化，不能访问元素，一般用来减少动态扩展次数

题目与应用（CCF2017）

题目描述：体育老师小明要将自己班上的学生按顺序排队。他首先让学生按学号从小到大的顺序排成一排，学号小的排在前面，然后进行多次调整。一次调整小明可能让一位同学出队，向前或者向后移动一段距离后再插入队列。

例如，下面给出了一组移动的例子，例子中学生的人数为8人。

0）初始队列中学生的学号依次为1, 2, 3, 4, 5, 6, 7, 8；

1）第一次调整，命令为“3号同学向后移动2”，表示3号同学出队，向后移动2名同学的距离，再插入到队列中，新队列中学生的学号依次为1, 2, 4, 5, 3, 6, 7, 8；

2）第二次调整，命令为“8号同学向前移动3”，表示8号同学出队，向前移动3名同学的距离，再插入到队列中，新队列中学生的学号依次为1, 2, 4, 5, 8, 3, 6, 7；

3）第三次调整，命令为“3号同学向前移动2”，表示3号同学出队，向前移动2名同学的距离，再插入到队列中，新队列中学生的学号依次为1, 2, 4, 3, 5, 8, 6, 7。

小明记录了所有调整的过程，请问，最终从前向后所有学生的学号依次是多少？

请特别注意，上述移动过程中所涉及的号码指的是学号，而不是在队伍中的位置。在向后移动时，移动的距离不超过对应同学后面的人数，如果向后移动的距离正好等于对应同学后面的人数则该同学会移动到队列的最后面。在向前移动时，移动的距离不超过对应同学前面的人数，如果向前移动的距离正好等于对应同学前面的人数则该同学会移动到队列的最前面。

题目与应用

输入格式

输入的第一行包含一个整数n，表示学生的数量，学生的学号由1到n编号。

第二行包含一个整数m，表示调整的次数。

接下来m行，每行两个整数p, q，如果q为正，表示学号为p的同学向后移动q，如果q为负，表示学号为p的同学向前移动-q。

输出格式

输出一行，包含n个整数，相邻两个整数之间由一个空格分隔，表示最终从前向后所有学生的学号。

样例输入

8

3

3 2

8 -3

3 -2

样例输出

1 2 4 3 5 8 6 7

代码实现：

```
#include <iostream>
#include <vector>
using namespace std;
vector<int> v;
int find(int x){
    for(int i=0;i<v.size();i++){
        if(v[i]==x)//遍历查找学号
        return i;//记录下该学号现在所处的位置
        }
    return -1;
}
int main(){
    int n,m,p,q;
    cin>>n>>m;
    for(int i=1;i<=n;i++){
        v.push_back(i);//注意1是填充到v[0]上的，注意下标和元素的区别
    }
    for(int i=1;i<=m;i++){
        cin>>p>>q;
        int flag=find(p);
        v.insert(v.begin() + flag + (q > 0 ? q + 1 : q), p);//如果q大于0那么q+1，否则q不变，在v.begin()+flag+(q > 0 ? q + 1 : q)位置插入元素p
        v.erase(v.begin() + flag + (q < 0 ? 1 : 0));//删除原来p所在位置上的元素p
    }
    for(int i=0;i<v.size();i++)
    cout<<v[i]<<" ";
    cout<<endl;
}
```

列表(list)

list概述：list是一种序列式容器，list中的数据元素是通过链表指针串连成逻辑意义上的线性表。STL中的链表是一个双向循环链表，由于链表的存储方式并不是连续的内存空间，因此链表list中的迭代器只支持前移和后移，属于双向迭代器。

list的数据结构：是一种物理存储单元上非连续的存储结构，数据元素的逻辑顺序是通过链表中的指针链接实现的。

list的特点：采用动态存储分配，不会造成内存浪费和溢出。链表执行插入和删除操作十分方便，修改指针即可，不需要移动大量元素。链表灵活，但是空间(指针域) 和 时间（遍历）额外耗费较大。

list基本操作（特色）：

1.不能随机访问

```
list<int> l;
l.push_back(10);l.push_back(20);l.push_back(30);
cout<<l.front()<<endl;
cout<<l.back()<<endl;
list<int>::iterator it = l.begin();
//it = it + 1;//错误，不可以跳跃访问，即使是+1
```

2.支持排序反转

```
list<int> l;l.push_back(10);l.push_back(20);l.push_back(30);
l.reverse();
l.sort();//默认从小到大
```

题目与应用（CCF2017）

题目描述：体育老师小明要将自己班上的学生按顺序排队。他首先让学生按学号从小到大的顺序排成一排，学号小的排在前面，然后进行多次调整。一次调整小明可能让一位同学出队，向前或者向后移动一段距离后再插入队列。

例如，下面给出了一组移动的例子，例子中学生的人数为8人。

0）初始队列中学生的学号依次为1, 2, 3, 4, 5, 6, 7, 8；

1）第一次调整，命令为“3号同学向后移动2”，表示3号同学出队，向后移动2名同学的距离，再插入到队列中，新队列中学生的学号依次为1, 2, 4, 5, 3, 6, 7, 8；

2）第二次调整，命令为“8号同学向前移动3”，表示8号同学出队，向前移动3名同学的距离，再插入到队列中，新队列中学生的学号依次为1, 2, 4, 5, 8, 3, 6, 7；

3）第三次调整，命令为“3号同学向前移动2”，表示3号同学出队，向前移动2名同学的距离，再插入到队列中，新队列中学生的学号依次为1, 2, 4, 3, 5, 8, 6, 7。

小明记录了所有调整的过程，请问，最终从前向后所有学生的学号依次是多少？

请特别注意，上述移动过程中所涉及的号码指的是学号，而不是在队伍中的位置。在向后移动时，移动的距离不超过对应同学后面的人数，如果向后移动的距离正好等于对应同学后面的人数则该同学会移动到队列的最后面。在向前移动时，移动的距离不超过对应同学前面的人数，如果向前移动的距离正好等于对应同学前面的人数则该同学会移动到队列的最前面。

题目与应用

输入格式

输入的第一行包含一个整数n，表示学生的数量，学生的学号由1到n编号。

第二行包含一个整数m，表示调整的次数。

接下来m行，每行两个整数p, q，如果q为正，表示学号为p的同学向后移动q，如果q为负，表示学号为p的同学向前移动-q。

输出格式

输出一行，包含n个整数，相邻两个整数之间由一个空格分隔，表示最终从前向后所有学生的学号。

样例输入

8

3

3 2

8 -3

3 -2

样例输出

1 2 4 3 5 8 6 7

代码实现：

```cpp
#include <iostream>
#include <list>
#include <algorithm>
using namespace std;

int main(){
    int m,n,p,q;
    cin>>n>>m;
    list<int> l;
    for(int i=1;i<=n;i++){
        l.push_back(i);
    }
    for(int i=1;i<=m;i++){
        cin>>p>>q;
        list<int>::iterator it=find(l.begin(),l.end(),p);//找元素p所在的位置
        list<int>::iterator it1=it;//用另一个迭代器记录p所在位置
        if(q>0){//向后插
            for(int i=0;i<=q;i++)//如4原本位置是4，然后要往后插3，它要移动4步到7后面
            it1++;
        }
        else
        for(int i=1;i<=-q;i++)//4要往前插2，它只需移动两步到2前面
        it1--;//说白了，看空格数
        l.insert(it1,p);
        l.erase(it);
    }
    //list<int>::iterator it2=l.begin();
    for(list<int>::iterator it2=l.begin();it2!=l.end();it2++){
        cout<<*it2<<" ";
    }
    cout<<endl;
}
```

双端队列(deque)

deque概述：deque容器为一个给定类型的元素进行线性处理，像向量一样，它能够快速地随机访问任一个元素，并且能够高效地插入和删除容器的尾部元素。

deque数据结构：具有类似队列和栈的性质。内部有个中控器，维护每段缓冲区中的内容，缓冲区中存放真实数据，中控器维护的是每个缓冲区的地址，使得使用deque时像一片连续的内存空间。

deque特色：对头部的插入删除速度回比vector快，迭代器也是支持随机访问的。

deque基本操作：

1.头插尾插以及删除

```
deque<int> d;
d.push_back(10);
d.push_back(20);
d.push_front(30);//头插
d.pop_back();
d.pop_front();
```

题目及其应用：

有5名选手：选手ABCDE，10个评委分别对每一名选手打分，去除最高分，去除评委中最低分，取平均分。创建五名选手，放到vector中，遍历vector容器。取出来每一个选手，执行for循环，可以把10个评分打分存到deque容器中。sort算法对deque容器中分数排序，去除最高和最低分，deque容器遍历一遍，累加总分，获取平均分。

```
class person{
    public:
        person(string name,int score){//构造函数
            this->m_name=name;
            this->m_score=score;
        }
        string m_name;
        int m_score;
};
void creatperson(vector<person>& v){
    string nameseed="ABCDE";
    for(int i=0;i<5;i++){
        string name="选手";
        name+=nameseed[i];//利用string的特性
        int score=0;
        person p(name,score);
        v.push_back(p);//存进去
    }
}
void setscore(vector<person>&v){
    for(vector<person>::iterator it = v.begin(); it != v.end(); it++){
        deque<int> d;
        for(int i=0;i<10;i++){
        int score=rand()%41+60;//60~100随机种子
        d.push_back(score);}
    sort(d.begin(),d.end());
```

```
    sort(d.begin(),d.end());
    d.pop_back();
    d.pop_front();
    int sum=0;
    for(deque<int>::iterator dit=d.begin();dit!=d.end();dit++){
        sum+=*dit;
    }
    int avg=sum/d.size();
    it->m_score=avg;
}
}

void show(vector<person>&v)
{
    for(vector<person>::iterator vit=v.begin();vit!=v.end();vit++)
    {
        cout << "姓名：  " << vit->m_name << " 平均分：  " << vit->m_score << endl;
    }
}

int main(){
    srand((unsigned int)time(NULL));//随机数种子
    vector<person> v;
    creatperson(v);
    setscore(v);
    show(v);
    system("pause");
    return 0;
}
```

栈(stack)

stack概述：stack是限定仅在一端进行插入或删除的操作的线性表，它只有一个出口。栈中只有顶端的元素才可以被外界使用，因此栈不允许有遍历行为。

stack数据类型：是一种先进后出的数据结构。

stack的特点：遵守“先进后出"，限定只能在栈顶进行插入和删除操作。基于数组的栈——以数组为底层数据结构时，通常以数组头为栈底，数组头到数组尾为栈顶的生长方向。基于单链表的栈——以链表为底层的数据结构时，以链表头为栈顶，便于节点的插入与删除，压栈产生的新节点将一直出现在链表的头部

stack基本操作：

```
stack<int> s;
s.push(10);s.push(20);s.push(30);
while (!s.empty()) {
cout << "栈顶元素为：  " << s.top() << endl;
s.pop();
}
cout << "栈的大小为：" << s.size() << endl;
}
```

题目及其应用：P6704
题目背景:Darko 有一个想象的外星朋友，他有十亿根手指。外星人快速拿起吉他，在网上找到一段简单的旋律并开始弹奏。这个吉他像寻常一样有六根弦，令其用 11 到 66 表示。每根弦被分成 PP 段，令其用 11 到 PP 表示。旋律是一串的音调，每一个音调都是由按下特定的一根弦上的一段而产生的（如按第 44 弦第 88 段）。如果在一根弦上同时按在几段上，产生的音调是段数最大的那一段所能产生的音调。
例：对于第 33 根弦，第 55 段已经被按，若你要弹出第 77 段对应音调，只需把按住第 77 段，而不需放开第 55 段，因为只有最后的一段才会影响该弦产生的音调(在这个例子中是第 77 段)。类似，如果现在你要弹出第 22 段对应音调，你必须把第 55 段和第 77 段都释放。请你编写一个程序，计算外星人在弹出给定的旋律情况下，手指运动的最小次数。
题目描述:你有一个 6×P 的矩阵 A，初始状态皆为0。对于所有要求 (i,j)(i,j)
你需要满足要求：此时 Ai,j状态为 1。对于 Ai,j+k(k>0)状态为0。
你在满足要求的情况下需要求状态转换最小次数。
输入格式:第一行包含两个正整数 n,P。它们分别指旋律中音调的数量及每根弦的段数。
下面的 n 行每行两个正整数 i，j，分别表示能弹出对应音调的位置——弦号和段号，其为外星人弹奏的顺序。
输出格式
一个非负整数表示外星人手指运动次数最小值。

输入：

5 15
2 8
2 10
2 12
2 10
2 5

输出：7

```
int main(){
    int n,p,x,y,cnt=0;
    stack<int>s[7];//每一条弦都有一个栈
    cin>>n>>p;
    for(int i=0;i<n;i++){
        cin>>x>>y;
    while(s[x].size()>0&&s[x].top()>y)
    {cnt++;
    s[x].pop();}//当这个栈不为空，且先输入的值大于
    //当前输入的值那么要让大的值出栈，且cnt+1，持续到这个top小于
    if(s[x].size()>0){
            if(s[x].top()==y)//如果刚好是栈顶，也就是之前摁着这个弦，那就不用出栈
            continue;
            else{//如果没有就入栈
                cnt++;
                s[x].push(y);
            }
            }
            else{//如果size是空的，就直接入栈，cnt+1
                cnt++;
                s[x].push(y);
            }
    }
    cout<<cnt;
    return 0;
}
```

队列(queue)

queue概述：是一种特殊的线性表，有两个入口，允许从一端新增元素，从另一端移除元素。
queue数据结构：是一种先进先出（FIFO）的数据结构。
queue特点：队列容器允许从一端新增元素，从另一端移除元素，队列中只有队头和队尾才可以被外界使用，因此队列不允许有遍历行为，还有优先队列这一特殊模板类。
priority_queue<Type, Container, Functional>　其中Type 为数据类型， Container 为保存数据的容器，Functional 为元素比较方式。Container 必须是用数组实现的容器，比如 vector, deque 但不能用 list。STL里面默认用的是 vector。
priority_queue基本操作：

```
priority_queue<int,vector<int>,greater<int> >q;//升序队列
for(int i=0;i<n;i++){
cin>>a;
q.push(a);} //插入元素到队尾（并排序）
cout<<q.top();//输出最小的那个元素
```

题目及其应用：
在一个果园里，多多已经将所有的果子打了下来，而且按果子的不同种类分成了不同的堆。多多决定把所有的果子合成一堆。
每一次合并，多多可以把两堆果子合并到一起，消耗的体力等于两堆果子的重量之和。可以看出，所有的果子经过 n-1n−1 次合并之后， 就只剩下一堆了。多多在合并果子时总共消耗的体力等于每次合并所耗体力之和。
因为还要花大力气把这些果子搬回家，所以多多在合并果子时要尽可能地节省体力。假定每个果子重量都为 11 ，并且已知果子的种类 数和每种果子的数目，你的任务是设计出合并的次序方案，使多多耗费的体力最少，并输出这个最小的体力耗费值。
例如有 33 种果子，数目依次为 11 ， 22 ， 99 。可以先将 11 、 22 堆合并，新堆数目为 33 ，耗费体力为 33 。接着，将新堆与原先的第三堆合并，又得到新的堆，数目为 1212 ，耗费体力为 1212 。所以多多总共耗费体力 =3+12=15=3+12=15 。可以证明 1515 为最小的体力耗费值。
输入格式
共两行。第一行是一个整数 n(1≤n≤10000) ，表示果子的种类数。
第二行包含n个整数，用空格分隔，第i个整数ai(1≤ai≤20000) 是第i种果子的数目。
输出格式
一个整数，也就是最小的体力耗费值。输入数据保证这个值小于 2^31。

输入：	输出：
3 1 2 9	15

代码及其实现：

```
using namespace std;
/* run this program using the console pauser or add your own g
int main(int argc, char** argv) {
    int a,b,n,p=0;
    cin>>n;
    priority_queue<int,vector<int>,greater<int> >q;//升序队列
    for(int i=0;i<n;i++){
        cin>>a;
        q.push(a);//插入元素到队尾（并排序）
    }
    int sum=0;
    for(int i=0;i<n-1;i++){
       b=q.top();//访问队头元素
       q.pop();//弹出队头元素，此时第二项成为队头元素
       b+=q.top();//b成为前两项之和
       q.pop();//把成为队头元素的第二项弹出
       sum+=b;//确保每次sum都是加上最小的
       q.push(b);//把b插入并排序
    }
    cout<<sum;
    return 0;
}
```

谢谢.

THANKS FOR WATCHING