# [[ACM训练] 算法初级 之 数据结构 之 栈stack+队列queue (基础+进阶)](http://www.cnblogs.com/yushuo1990/p/5995051.html)

首先，这两个数据结构都是比较常用的，在标准库中都有对应的结构能够直接使用，所以第一个阶段应该是先学习直接来使用，下一个阶段再去探究具体的实现，以及对基本结构的改造！

C++标准库中的基本使用方法：

栈： #include<stack>

定义栈，以如下形式实现： stack<Type> s;

其中Type为数据类型（如 int，float,char等）

常用操作有：

s.push(item); 　　    //将item压入栈顶  
**s.pop();** 　　　　 　 //删除栈顶的元素，但不会返回  
**s.top();** 　　　　　  //返回栈顶的元素，但不会删除，**在出栈时需要进行两步，**即先top()获得栈顶元素，再pop()删除栈顶元素****  
s.**size()**;　　　　　　//返回栈中元素的个数  
s.empty();　　　　  //检查栈是否为空，如果为空返回true，否则返回false

最基本的用法就是：

stack<int> st;---------------------------------->栈

int first=1;

st.push(first);//入栈1

int second=2;

st.push(second);//入栈2

**first=st.top();//first变成了2**

**st.pop();//出栈**

 队列：#include<queue>

queue<int> q; //定义一个 int 型的队列 ---------------------------------->队列  
q.empty()//如果队列为空返回true，否则返回false   
q.size() //返回队列中元素的个数

q.push() //在队尾压入新元素  
**q.front() //返回队首元素的值，但不删除该元素**   
q.pop() //删除队**列首**元素但不返回其值   
q.back()//返回队列尾元素的值，但不删除该元素

另外在队列中还有优先队列：priority\_queue<.........>--------------------------->优先队列

优先队列是我们比较不熟悉的一种结构，下面整体上做一个简单了解：

优先队列是队列的一种，不过它可以按照自定义的一种方式（数据的优先级）来对队列中的数据进行动态的排序，每次的push和pop操作，队列都会动态的调整，以达到我们预期的方式来存储。

例如：我们常用的操作就是对数据排序，优先队列默认的是**数据大的优先级高**，所以我们无论按照什么顺序push一堆数，最终在队列里总是top出最大的元素。

所以，优先队列在一些**定义了权重的地方**很有用，priority\_queue特别之处在于，允许**用户为队列中存储的元素设置优先级**。这种队列不是直接将新元素放置在队列尾部，而是放在比它优先级低的元素前面。

priority\_queue<int> bigque;//最大优先

priority\_queue<int, vector<int> ,greater<int> > smallque;//最小优先

要包含头文件：

#include<queue>

优先队列支持的操作有：

q.empty() 　　 //如果队列为空，则返回true，否则返回false  
q.size() 　　　 //返回队列中元素的个数  
q.pop()　　　 //删除队首元素，但不返回其值  
**q.top() 　　 //返回具有最高优先级的元素值，但不删除该元素，注意与传统队列的不同以及和栈的相同点**  
q.push(item) //**在基于优先级的适当位置插入新元**

Poj1338 Ugly Numbers

|  |  |  |
| --- | --- | --- |
| **Time Limit:** 1000MS |  | **Memory Limit:** 10000K |
| **Total Submissions:** 25576 |  | **Accepted:** 11181 |

**Description**

Ugly numbers are numbers whose only prime factors are 2, 3 or 5. The sequence   
1, 2, 3, 4, 5, 6, 8, 9, 10, 12, ...   
shows the first 10 ugly numbers. By convention, 1 is included.   
Given the integer n,write a program to find and print the n'th ugly number.

**Input**

Each line of the input contains a postisive integer n (n <= 1500).Input is terminated by a line with n=0.

**Output**

For each line, output the n’th ugly number .:Don’t deal with the line with n=0.

**Sample Input**

1

2

9

0

**Sample Output**

1

2

10

题意：定义这样一组数据，这组数据只由2，3，5互乘得到并且是递增序。让你输出第n大的数。N<=1500要注意的是1也算在里面

题解：首先定义一个最小优先队列，一个jg判断数组，一个map输出数组，将2，3，5入队。到这里初始化就已经结束了，接下来的步骤就是解题过程，依次从队首取出元素，计入map输出数组。比如说第一个2，，把它放在map[1]的位置，然后计算2\*2，2\*3，2\*5，将计算后的数值和jg判断数组比较，如果对应数据为0，表明队内无重复元素，此时将其入队，并将对应数据置为1，如果对应数据为1，则跳过，比较下一个数据。这样循环直到n==1500结束。

Poj2442 Sequence

|  |  |  |
| --- | --- | --- |
| **Time Limit:** 6000MS |  | **Memory Limit:** 65536K |
| **Total Submissions:** 10249 |  | **Accepted:** 3364 |

Description

Given m sequences, each contains n non-negative integer. Now we may select one number from each sequence to form a sequence with m integers. It's clear that we may get n ^ m this kind of sequences. Then we can calculate the sum of numbers in each sequence, and get n ^ m values. What we need is the smallest n sums. Could you help us?

Input

The first line is an integer T, which shows the number of test cases, and then T test cases follow. The first line of each case contains two integers m, n (0 < m <= 100, 0 < n <= 2000). The following m lines indicate the m sequence respectively. No integer in the sequence is greater than 10000.

Output

For each test case, print a line with the smallest n sums in increasing order, which is separated by a space.

Sample Input

1

2 3

1 2 3

2 2 3

Sample Output

3 3 4

题意：给你n\*m的矩阵，每行取一个数组成n\*n个序列，求这些序列前n个最小序列和并输出

题解：定义一个a[][]存输入的数据，一个map[]数组存待输出的数据，一个最大优先队列存操作数据。先将a[1]从小到大排序后存入map，然后取a[2][1]和map相加后入队，随后a[2]依次加上map中的数据后与最大队列队头进行比较，如果a[2]+map[i]的数据小于队中的队首，则队首出队，a[2]+map[i]入队，若大于队首，则跳出循环进行a[3]的比较。循环上述操作直到a[m][n].

**Poj1442 Black Box**

|  |  |  |
| --- | --- | --- |
| Time Limit: 1000MS |  | Memory Limit: 10000K |
| Total Submissions: 14042 |  | Accepted: 5726 |

Description

Our Black Box represents a primitive database. It can save an integer array and has a special i variable. At the initial moment Black Box is empty and i equals 0. This Black Box processes a sequence of commands (transactions). There are two types of transactions:   
  
ADD (x): put element x into Black Box;   
GET: increase i by 1 and give an i-minimum out of all integers containing in the Black Box. Keep in mind that i-minimum is a number located at i-th place after Black Box elements sorting by non- descending.   
  
Let us examine a possible sequence of 11 transactions:   
  
Example 1

N Transaction i Black Box contents after transaction Answer   
 (elements are arranged by non-descending)   
1 ADD(3) 0 3   
2 GET 1 3 3   
3 ADD(1) 1 1, 3   
4 GET 2 1, 3 3   
5 ADD(-4) 2 -4, 1, 3   
6 ADD(2) 2 -4, 1, 2, 3   
7 ADD(8) 2 -4, 1, 2, 3, 8   
8 ADD(-1000) 2 -1000, -4, 1, 2, 3, 8   
9 GET 3 -1000, -4, 1, 2, 3, 8 1   
10 GET 4 -1000, -4, 1, 2, 3, 8 2   
11 ADD(2) 4 -1000, -4, 1, 2, 2, 3, 8

It is required to work out an efficient algorithm which treats a given sequence of transactions. The maximum number of ADD and GET transactions: 30000 of each type.   
  
  
Let us describe the sequence of transactions by two integer arrays:   
  
  
1. A(1), A(2), ..., A(M): a sequence of elements which are being included into Black Box. A values are integers not exceeding 2 000 000 000 by their absolute value, M <= 30000. For the Example we have A=(3, 1, -4, 2, 8, -1000, 2).   
  
2. u(1), u(2), ..., u(N): a sequence setting a number of elements which are being included into Black Box at the moment of first, second, ... and N-transaction GET. For the Example we have u=(1, 2, 6, 6).   
  
The Black Box algorithm supposes that natural number sequence u(1), u(2), ..., u(N) is sorted in non-descending order, N <= M and for each p (1 <= p <= N) an inequality p <= u(p) <= M is valid. It follows from the fact that for the p-element of our u sequence we perform a GET transaction giving p-minimum number from our A(1), A(2), ..., A(u(p)) sequence.

Input

Input contains (in given order): M, N, A(1), A(2), ..., A(M), u(1), u(2), ..., u(N). All numbers are divided by spaces and (or) carriage return characters.

Output

Write to the output Black Box answers sequence for a given sequence of transactions, one number each line.

Sample Input

7 4

3 1 -4 2 8 -1000 2

1 2 6 6

Sample Output

3

3

1

2

题意：有add和get两种操作，add：将指定值放入序列，get：输出序列中第i大的值，i初始为0，每次get操作前++.上面的input为简化输入 7代表add次数也就是序列长度，4代表get次数也就是输出的个数。PS:get是非降序输入

题解：一个a[]数组，暂时存放add数组，一个最小优先队列存放add数组，一个tail变量。初始化结束。之后遍历get。比如第一个数1循环a[]至1下标，将a[1]入队，循环结束后输出队首元素,tail置为1，之后再访问get的第二个数据2，将a[tail]-a[2]入队，进行（2-tail)次队首出队操作后输出队首元素。之后的操作和上面一样。

**POJ：1338 2442 1442**