优先队列

优先队列

刷新

题目描述

普通的队列（queue）是一种先进先出的数据结构，元素在队列尾追加，从队列头删除。而在优先队列（priority_queue）中，元素被赋予优先级，可以用不超过 $O(\log n)$ 的时间复杂度插入一个元素、访问优先级最高的元素、或删除优先级最高的元素，具有最高级先出（first in, largest out）的行为特征，可以高效地动态维护序列最值。

人们常用堆这种数据结构来实现高效的优先队列。但在这题里，我们**并不需要你去实现高效的算法**，只需要你去模拟优先队列的行为。具体来说，你需要实现一个模板类 `PriorityQueue<T>(AbstractCompare<T>*)`，其中 `T` 为任意类型，`AbstractCompare<T>` 为抽象比较类。规定较小的元素为优先队列中优先级较高的元素。也就是说，每次删除时都是删除队列中最小的元素。

`PriorityQueue` 需要实现以下几种功能：

- 插入元素
- 删除队列中的最小值
- 查询队列中的最小值
- 查询队列中元素个数
- 查询队列是否为空

同时为了测验你的模板类能够应用于自定义类型，我们给出了 `Point` 对象。在 `T=Point` 的情况下，同样需要你完成以上操作。

数据分为四种类型，由输入的 `type` 控制。

- 类型 1：优先队列中存放 `int` 类型元素，比较方式为按数值比较；
- 类型 2：优先队列中存放 `int` 类型元素，比较方式为按数字和比较，如果数字和一样就按数值比较；举例：98 的数字和是 $9+8=17$，123 的数字和是 $1+2+3=6$。
- 类型 3：优先队列中存放 `Point` 类型元素，比较方式为先比较第一维坐标 `x` 的大小，如果 `x` 一样再比较 `y` 的大小。**请将该比较实现为 Point 类的默认大小关系（即重载 `operator<`）**；
- 类型 4：优先队列中存放 `Point` 类型元素，比较方式为按到原点 $(0,0)$ 的欧式距离比较，如果距离一样就按默认大小（即类型 3）关系比较。

请阅读代码，并实现 `priority_queue.h` 和 `compare.h` 中定义的接口。

本题并不考察插入、删除元素时的时间性能，你只需正确实现对应功能即可。

文件下载地址：下载链接 (/staticdata/1778.9u3ddWvwRWzIMBRA.pub/ELKhy53NQ1E8wKut.download_hw4_p4.zip/download_hw4_p4.zip)

提示

对于每种数据，你需要继承 `AbstractCompare` 类实现对应的子类，并在 `PriorityQueue` 中使用你实现的子类来完成数据的比较。

输入说明

输入的第一行包含一个正整数 `type`，表示数据种类，为上文 4 种类型之一；

输入的第二行包含一个正整数 n，表示操作数量；

接下来 n 行，每行先输入一个正整数 `op`，表示操作种类，

- $\texttt{op}=1$：再输入一个与维护类型相同类型的元素 `element`，并插入到队列中；
- $\texttt{op}=2$：若队列非空则删除队列中最小值；
- $\texttt{op}=3$：若队列非空则输出队列中最小值，否则输出 `invalid operation`；
- $\texttt{op}=4$：输出队列中元素个数。

输入数据保证任意两个元素不“相等”，也就是一定存在大小关系。

输出说明

对每个 $\texttt{op}=3,4$ 的操作输出一行，为对应的输出。

在所有操作结束后，依次输出并删除队列中所有元素（从小到大）。

对输入操作的实现都已包含在 `main.cpp` 中，如果你正确地实现了功能接口即可得到正确的输出。

样例

输入样例 1

```
1
12
1 5
4
1 12
3
2
3
2
3
2
1 6
1 24
4
```

输出样例 1

```
1
5
12
invalid operation
2
------------input finished------------
6
24
```

解释：

```
1                       // type=1
12                      // 共12个操作
1 5                     // 插入5，队列中元素为：5
4                       // 查询大小，输出1
1 12                    // 插入12，队列中元素为(从小到大)：5,12
3                       // 查询最小值，输出5
2                       // 删除最小值，队列中元素为：12
3                       // 查询最小值，输出12
2                       // 删除最小值，队列为空
3                       // 队列为空，输出"invalid operation"
2                       // 队列为空，不删除
1 6                     // 插入6，队列中元素为：6
1 24                    // 插入24，队列中元素为：6,24
4                       // 查询大小，输出2
```

输入样例 2

```
2
12
1 5
4
1 12
3
2
3
2
3
2
1 6
1 24
4
```

输出样例 2

```
1
12
5
invalid operation
2
------------input finished------------
6
24
```

解释：注意大小关系的变化

输入样例 3

```
3
9
1 0 0
1 1 0
1 1 1
1 0 1
1 -1 1
1 -1 0
1 -1 -1
1 0 -1
1 1 -1
```

输出样例 3

```
------------input finished------------
-1 -1
-1 0
-1 1
0 -1
0 0
0 1
1 -1
1 0
1 1
```

输入样例 4

```
4
9
1 0 0
1 1 0
1 1 1
1 0 1
1 -1 1
1 -1 0
1 -1 -1
1 0 -1
1 1 -1
```

输出样例 4

```
------------input finished------------
0 0
-1 0
0 -1
0 1
1 0
-1 -1
-1 1
1 -1
1 1
```

数据规模和约定

本题共设置 4 个子任务，各占 25 分，分别对应 $\texttt{type} = 1, 2, 3, 4$。

对于所有测试点，保证 $1 \leq n \leq 100$。

优先队列中存放 `int` 类型元素时，保证元素在 $[0, 100]$ 范围内；

优先队列中存放 `Point` 类型元素时，保证两维坐标的绝对值在 $[0, 100]$ 范围内；

保证输入的所有元素在对应的大小比较规则下不“相等”。

提交格式

请不要修改 `main.cpp`，`abstract_compare.h`，`point.h`，`point.cpp` 和 `Makefile`。

我们会将你提交的文件和我们预先设置好的 `main.cpp`，`abstract_compare.h`，`point.h`，`point.cpp`，`Makefile` 一起编译运行。

请将你的文件打包成一个 zip 压缩包并上传。注意：你的文件应该在压缩包的根目录下，而不是压缩包的一个子文件夹下，换而言之，解压你提交的压缩包后，应该直接得到你提交的 cpp 文件（或其他代码文件），而不是一个包含它们的文件夹。

语言和编译选项

#	名称	编译器	额外参数	代码长度限制
0	oop_custom	make		65536 B

递交历史

#	状态	时间

递交答案

语言和编译选项 oop_custom

```
1
```

提交

文件请拖入编辑器中，或 上传文件