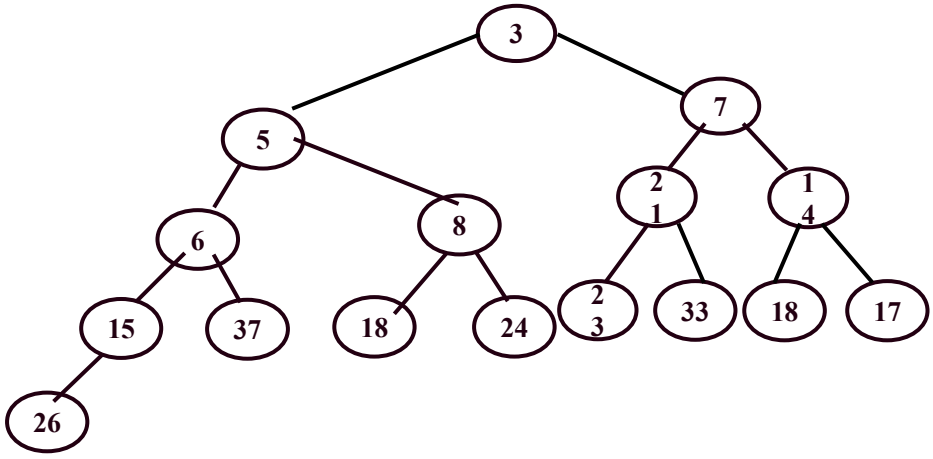
设计并实现一个基于最小化堆的的整型的优先级队列。即，队列元素为整型，数值越小，优先级越高。这个类除了实现队列的基本操作外，还需要实现下列功能：

⑴ int findMin（x）：找出优先级值大于指定元素 x 的最小元素，并返回它的下标。例如，在简答题 6 的二叉堆中找出大于 9 的最小元素。这个元素应该是 12，12 的下标为 4，所以返回 4。

⑵ decreaseKey（i，value）：将第 i 个结点的优先级值减少 value。如，在简答题 6 的二叉堆中执行 decreaseKey（4，7），结果如下图所示



【解】我们可以在代码清单 6-1 中的优先级队列类中增加两个公有的成员函数完成 findMin 和 decreaseKey 的操作。优先级队列类的定义如代码清单 6-28 所示。

代码清单 6-28 优先级队列类的定义

```
1.  template <class T>
2.  class priorityQueue {
3.     public:
4.        priorityQueue(int capacity = 100) ;
5.        priorityQueue(const T data[], int size);
6.        ~priorityQueue();
7.
8.        bool isEmpty() const;
9.        void enQueue(const T& x) ;
10.       T deQueue() ;
11.       T getHead() const ;
12.
13.       int findMin(T x) {    // 找出大于等于 x 的最小元素
14.          T Min;
15.          int ID = -1;
16.
17.          for(int i = 1; i <= currentSize; ++i)
18.             if(array[i] >= x && (Min == -1 || array[i] < Min)) {
19.                Min = array[i];
```

```
20.                 ID = i;
21.               }
22.           return ID;
23.       }
24.
25.       void decreaseKey(int i, T value) {  // 将堆中第 i 个结点的值减少 value
26.           T x;
27.           int hole;
28.
29.           array[hole = i] -= value;      // 将第 i 个元素减少 value
30.           for(x = array[i]; hole>1 && x < array[hole/2]; hole /= 2)//过滤
31.               array[hole] = array[hole / 2];
32.           array[hole] = x;
33.       }
34.
35.    private:
36.       int currentSize;
37.       T *array;
38.       int maxSize;
39.
40.       void doubleSpace() ;
41.       void buildHeap() ;
42.       void percolateDown(int hole);
43. };
```

相比于代码清单 6-1 中的优先级队列类，代码清单 6-26 中的优先级队列类增加了两个公有的成员函数 findMin 和 decreaseKey，其它函数的含义、定义和实现与代码清单 6-1 中的优先级队列类完全相同。下面我们介绍一下新增加的两个函数。

findMin 函数在优先级队列中找出大于等于 x 的最小元素所在的下标值。它采用最简单的顺序检查的方法，在数组中找出大于等于 x 的最小元素所在的下标值，并返回此下标值。

decreaseKey 函数有两个参数 i 和 value，该函数将存储在下标 i 中的元素值减小 value。将存储在下标 i 中的元素值减小 value 后将会破坏堆的有序性。该元素值可能小于它的祖先结点，但以该元素为根的子堆仍然保持堆的有序性。这个情况类似于入队时的情况。我们可以对这个结点实施向上过滤的过程，让它移动到合适的位置。在它往上移动的过程中，它的祖先结点会往下移动，成为它原先的子堆的根结点。由于它的祖先结点都比它的子孙结点小，所以移动后不会违反堆的有序性。

总结一下，decreaseKey 函数的实现由两个过程组成。先将下标值为 i 的元素值减小 value，然后对此元素执行向上过滤。