最少背包问题：假设有许多盒子，每个盒子能保存的总重量为 1.0。有 N 个项 $i_1,i_2,\cdots,i_N$，它们的重量分别是 $w_1,w_2,\cdots,w_N$。目的是用尽可能少的盒子放入所有的项，任何盒子的重量不能超过他的容量。例如，如果想的重量为 0.4, 0.4, 0.6 和 0.6，用两个盒子就能解决。按如下策略解决此问题：按给定的次序扫描每一个项，把每一个项放入能够容纳他而不至于溢出的最满的盒子。用第 7 题的优先级队列选择要装入的盒子

【解】题目要求用贪婪法解决。它不一定能得到最优的答案，只能得到较好的方案。首先定义一个优先级队列，队列的元素是一个背包中的剩余容量。对于 n 件物品，最多一共需要 n 个背包。将所有背包的剩余容量置为 1.0，入队。之后每一次遇到一件物品，按照题示规则，放入第一个能够容纳它的最满的背包。所以需要找到能装入物品的盒子所在队列中的位置，这可以用 findMin 完成，然后将它的剩余容量减掉物品的重量，这可以用 decreaseKey 实现。每装入一件物品，都要检查队首元素的容量是否为 0。如果为 0，表示背包中不能再装任何物品，没必要再留在队列中，将它出队，同时记录装满的背包的个数。所有物品装入后，将队列清空。在清空的过程中，统计所有剩余容量不为 1.0 的盒子数，最后输出所有不为空的背包的个数。这个过程如代码清单 6-29。

代码清单 6-29 最小背包问题

```
1.  int main(){
2.      int n, num = 0;                 // n：物品数量，num：所需的背包数
3.      double *ar;                     // 存放 n 件物品的重量
4.      priorityQueue<double> myQueue; // 以背包的剩余容量为优先级组成的背包队列
5.
6.      // 初始化
7.      cout << "请输入一共多少件物品：\n";
8.      cin >> n;
9.      ar = new double[n];
10.     cout << "请依次输入每件物品的数量：\n";
11.     for(int i = 0 ; i < n ; ++i){       // 最坏情况下背包数与物品数相同
12.         cin >> ar[i];
13.         myQueue.enQueue(1.0);           // 背包的初始容量都是 1.0
14.     }
15.
16.     for(i = 0 ; i < n ; ++i){           // 逐个放入 n 件物品
17.         myQueue.decreaseKey(myQueue.findMin(ar[i]),ar[i]);
18.         while(myQueue.getHead() == 0){ ++num;myQueue.deQueue(); }
19.     }
20.     while(!myQueue.isEmpty())           // 检查用到的背包数
21.         if (myQueue.deQueue() < 1.0) ++num ;
22.
23.     cout << "按照此种给定顺序和贪心算法，得到需要的背包总数为：" << num << endl;
24.
25.     delete [] ar;
```

```
26.         return 0;
27. }
```

对于题目中给出的数据，该程序输出的方案是需要 3 个背包，背包中的物品重量分别是 0.8, 0.6，0.6。可见贪婪法不一定能得到最优的方案。