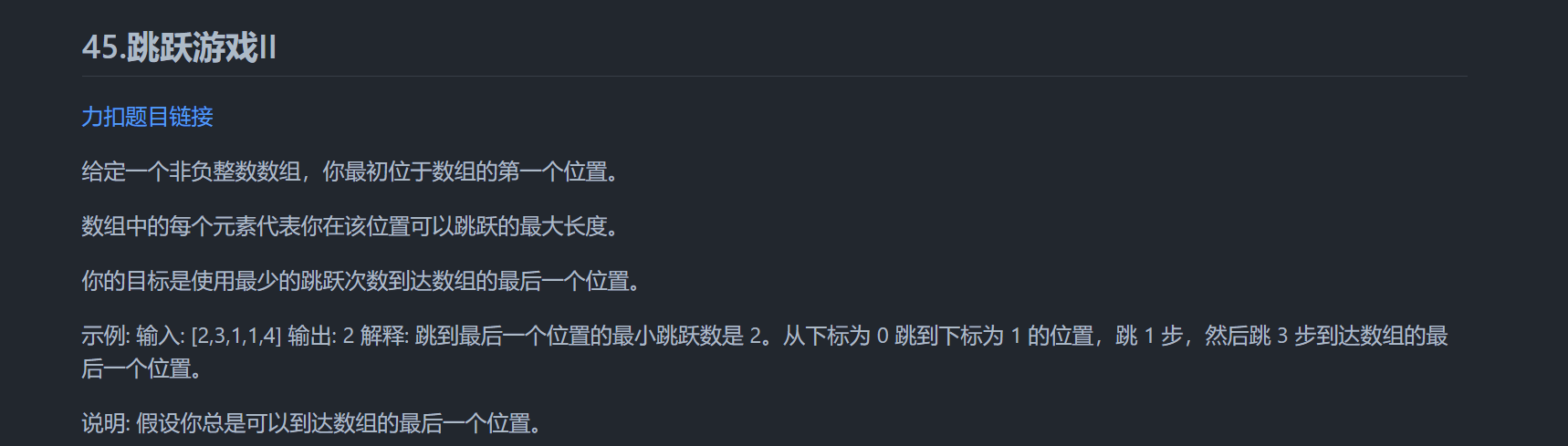
# 贪心算法（强化）

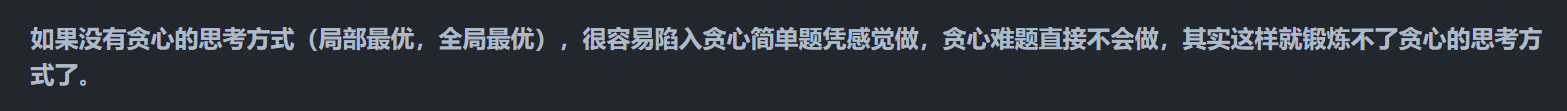
**2021/10/19**



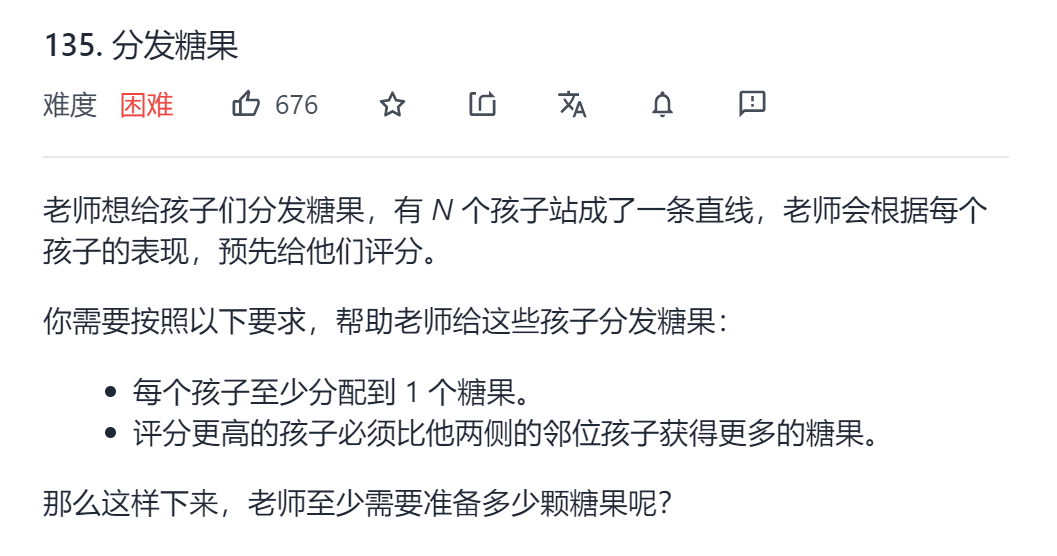
## 45:跳跃游戏

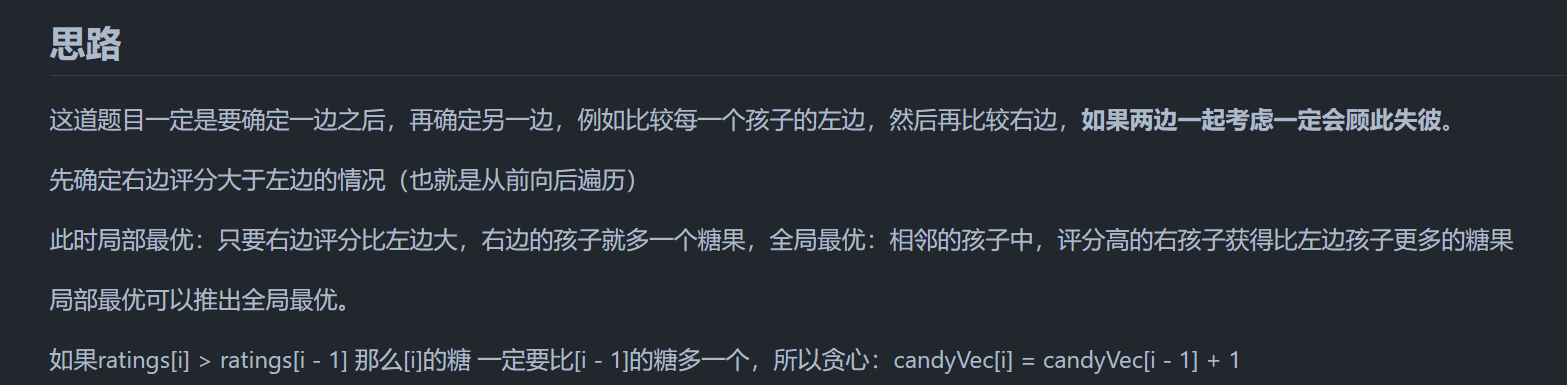


**贪心思路：每一步尽可能的走的远；**



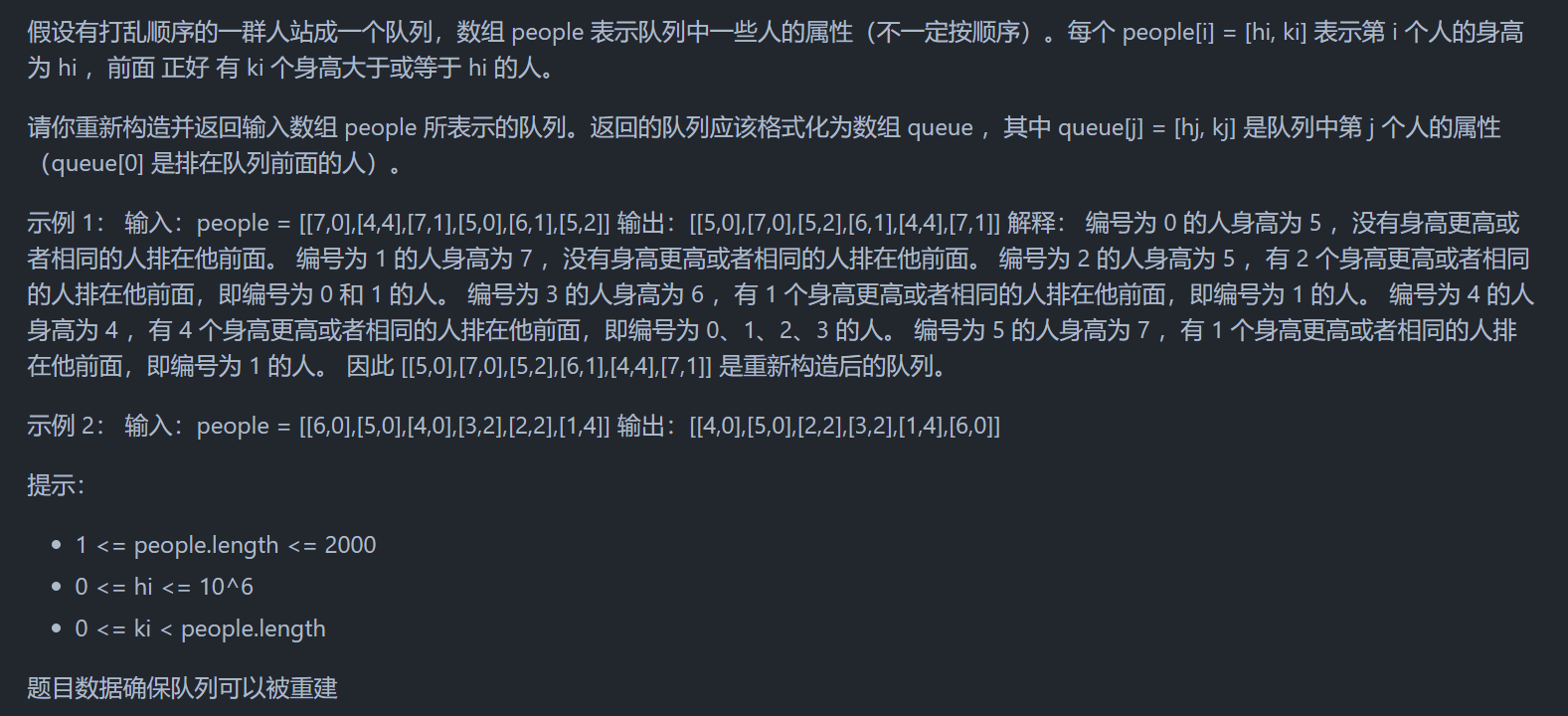
## 135:分发糖果（重点理解局部最优和全局最优）





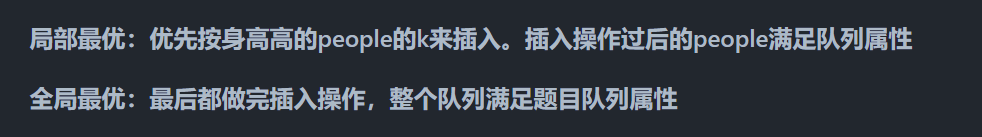
class Solution {  
public:  
 int candy(vector<int>& ratings) {  
 vector<int> candyVec(ratings.size(), 1);  
 // 从前向后  
 for (int i = 1; i < ratings.size(); i++) {  
 if (ratings[i] > ratings[i - 1]) candyVec[i] = candyVec[i - 1] + 1;  
 }  
 // 从后向前  
 for (int i = ratings.size() - 2; i >= 0; i--) {  
 if (ratings[i] > ratings[i + 1] ) {  
 candyVec[i] = max(candyVec[i], candyVec[i + 1] + 1);  
 }  
 }  
 // 统计结果  
 int result = 0;  
 for (int i = 0; i < candyVec.size(); i++) result += candyVec[i];  
 return result;  
 }  
};

## 406.根据身高重建队列（两个维度的贪心）



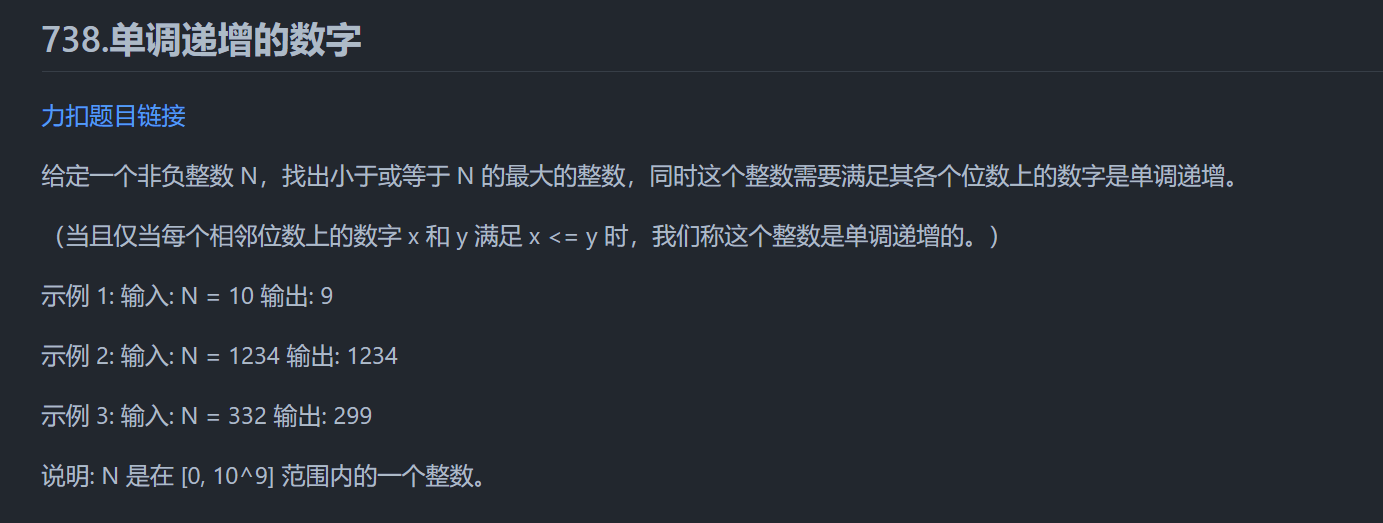
**本题有两个维度，h和k，看到这种题目一定要想如何确定一个维度，然后在按照另一个维度重新排列。**

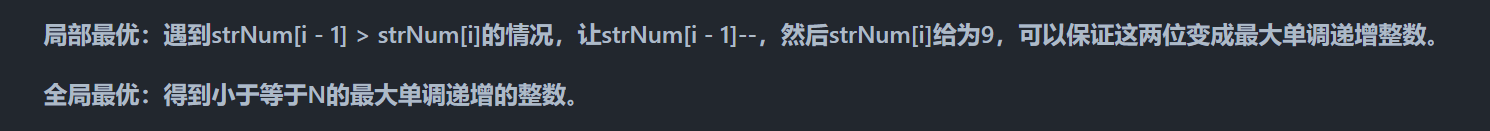
**如果两个维度一起考虑一定会顾此失彼。**



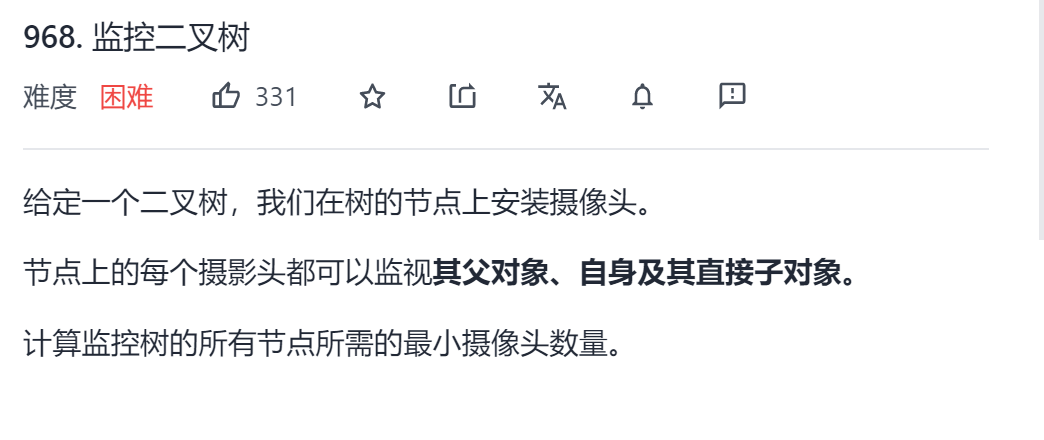
// 版本二  
class Solution {  
public:  
 // 身高从大到小排（身高相同k小的站前面）  
 static bool cmp(const vector<int> a, const vector<int> b) {  
 if (a[0] == b[0]) return a[1] < b[1];  
 return a[0] > b[0];  
 }  
 vector<vector<int>> reconstructQueue(vector<vector<int>>& people) {  
 sort (people.begin(), people.end(), cmp);  
 list<vector<int>> que; // list底层是链表实现，插入效率比vector高的多  
 for (int i = 0; i < people.size(); i++) {  
 int position = people[i][1]; // 插入到下标为position的位置  
 std::list<vector<int>>::iterator it = que.begin();  
 while (position--) { // 寻找在插入位置  
 it++;  
 }  
 que.insert(it, people[i]);  
 }  
 return vector<vector<int>>(que.begin(), que.end());  
 }  
};

## 738：单调递增的数字





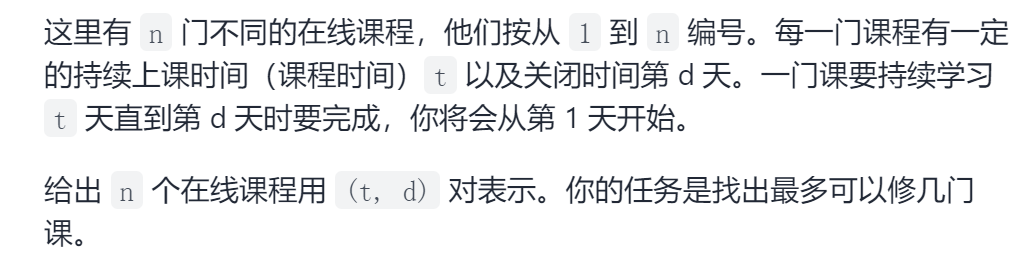
## 968:监控二叉树（贪心）

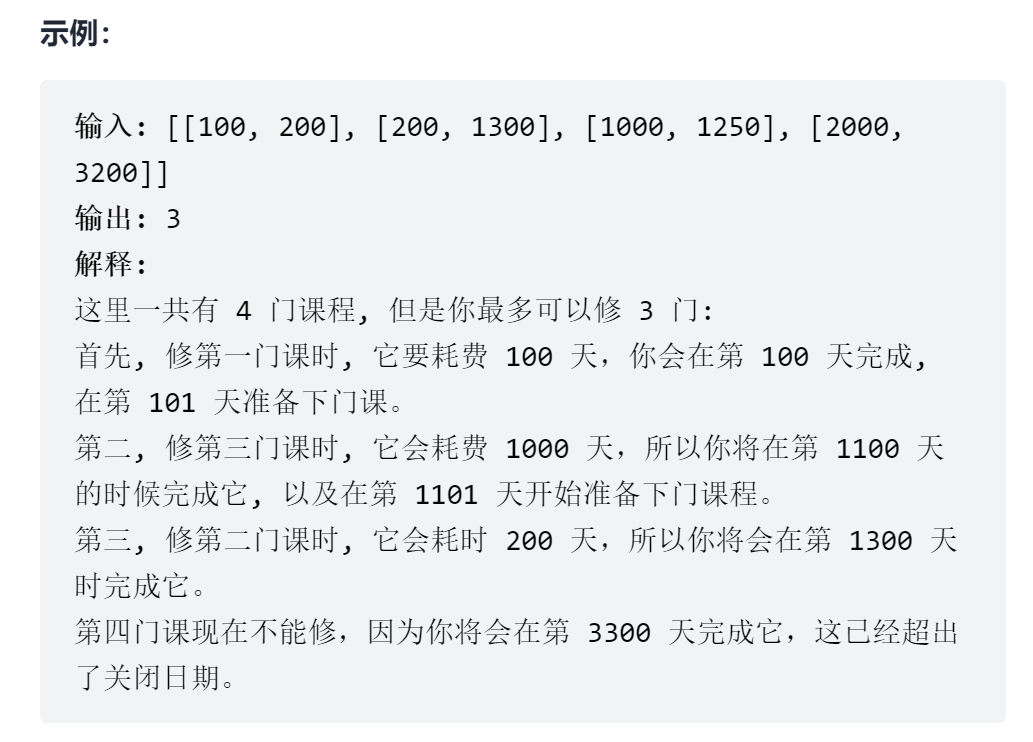


/\*\*  
 \* Definition for a binary tree node.  
 \* struct TreeNode {  
 \* int val;  
 \* TreeNode \*left;  
 \* TreeNode \*right;  
 \* TreeNode() : val(0), left(nullptr), right(nullptr) {}  
 \* TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}  
 \* TreeNode(int x, TreeNode \*left, TreeNode \*right) : val(x), left(left), right(right) {}  
 \* };  
 \*/  
class Solution {  
private:  
 int res;  
 int dfs(TreeNode\*root){  
 //返回0：表示空结点  
 //返回1：表示子结点  
 //返回2:表示装了监控的子节点  
 if(!root)return 0;  
 int left,right;  
 left=dfs(root->left);  
 right=dfs(root->right);  
 if(left==1||right==1){  
 //左右结点中至少有一个是没有装摄像头的子节点  
 ++res;  
 return 2;  
 }  
 //左右结点中至少有一个结点装了摄像头  
 if(left==2||right==2)return 0;  
 //if(left==0&&right==0)return 1;  
 //剩下的情况：没有子节点  
 return 1;  
  
 }  
public:  
 int minCameraCover(TreeNode\* root) {  
 //从底部向上进行贪心  
 //贪心：如果一个结点有一个子节点，那么这个结点上就一定要安装一个摄像头（局部最优）  
 //安装了摄像头结点看成空结点，继续向上贪心  
 res=0;  
 //根节点特殊处理  
 int l=dfs(root->left);  
 int r=dfs(root->right);  
 //根节点是唯一一个没有父节点的结点  
 //所以当左右都为空时，根节点需要装上摄像头  
 if(l==1||r==1||(l==0&&r==0))++res;  
 return res;  
 }  
};

## [630. 课程表 III](https://leetcode-cn.com/problems/course-schedule-iii/)

### 题目描述：





## 题目分析：（贪心+优先队列）

1. **这道题是一道关于贪心的题；这道题的贪心没有那么的显而易见，而是需要优化的贪心。**
2. **首先课程的终止时间表示一个课程的紧急程度；这里的一个贪心点就是：我们应该选择终止时间最早的课程**
3. **但是在这个贪心点上并不能保证最后得出的答案一定是最优的：因为终止时间早的课程也有可能花费更多的时间；**
4. **贪心的优化就是，如果已经选择的课程中有比现在要选的课程中更费时的课程，那么取代之！！！**

class Solution {  
private:  
 bool static cmp(vector<int>&a,vector<int>&b){  
 if(a[1]==b[1])return a[0]<b[0];  
 return a[1]<b[1];  
 }  
public:  
 int scheduleCourse(vector<vector<int>>& courses) {  
 //贪心算法+数据结构（优先队列）  
 //优先队列-自定义排序方法  
 //先学习结束时间最早的课，如过之后有课学习不了，判断是否可以将该可和最耗时的课程置换  
 sort(courses.begin(),courses.end(),cmp);  
 priority\_queue<int>que;  
 int time=0;//表示时间和  
 int numcour=0;//表示已经修的课程  
 int n=courses.size();  
 for(int i=0;i<n;i++){  
 if(courses[i][0]>courses[i][1])continue;  
 if(time+courses[i][0]<=courses[i][1]){  
 time+=courses[i][0];  
 ++numcour;  
 que.push(courses[i][0]);  
 }else{   
 //if(time-que.top()+courses[i][0]<=courses[i][1]){//这个条件并不能保证被替换掉的课程一定比该课程更费时间  
 if(!que.empty()&&que.top()>courses[i][0]){  
 time=time-que.top()+courses[i][0];  
 que.pop();  
 que.push(courses[i][0]);  
 }  
 }  
 }  
 return numcour;  
 }  
};

## 该题总结：

* **复杂一点的贪心往往需要在一个贪心点的基础之上进行优化！！！**
* **补充说明优先队列自定义排序：**
  1. //优先队列默认是从大到小排序的  
     priority\_queue<int> q1;//默认从大到小排序，整数中元素大的优先级高   
     //如果要改成从小到大排序，可以加上greater<类型>  
     priority\_queue<int,vector<int>,greater<int> >q1;  
       
     //自定义排序  
     struct cmp1{  
      bool operator()(int x,int y)  
      {  
      return x>y;//小的优先级高 ,从小到大排 //注意此处的比较与sort中数组排序中的比较相反  
      }  
     };   
     struct cmp2{  
      bool operator()(const int x,const int y)  
      {  
      return tmp[x]>tmp[y];  
      }  
     };   
     struct node{  
      int x,y;  
      friend bool operator<(node a,node b)  
      {  
      return a.x>b.x;//按x从小到大排   
      }  
     };  
     priority\_queue<int>q1;  
     priority\_queue<int,vector<int>,cmp1>q2;  
     priority\_queue<int,vector<int>,cmp2>q3;  
       
     ————————————————  
     版权声明：本文为CSDN博主「yanyanwenmeng」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。  
     原文链接：https://blog.csdn.net/yanyanwenmeng/article/details/78153192

# 贪心类题总结：

* + 1. **简单的贪心类的题可能就是一个维度或者就是不许要附加条件进行优化；只要找准一个贪心的点，就可以比较容易的想出来；但是实现起来可能并不是很容易**
    2. **比较难的贪心类的题可能就涉及到比较多的维度：这里的根据身高重建队列和课程表就是两个维度上的贪心。这类贪心的做法一般是在某个维度贪心的基础上用另一个维度进行优化；**
    3. **贪心题总体上来说是从局部最优推出整体最优！！！**