堆的应用

优先队列习题选讲1760,1134,1122, 2294, 3645, 1587

堆的应用0：逃亡 NKOI 1760

何老板抢劫了银行，他驾驶一辆卡车逃跑。卡车每行驶一公里就会消耗掉一升油。

何老板需要把车开到最近的一个小镇，那里有同伙接应。从卡车当前位置(银行)到小镇的这条路上，分布着N个加油站 ，何老板可以停下来加油，但是这条路上每个加油站的油量不超过100升，其中第i个加油站的油量为Gi。

警察正在追捕何老板，所以，何老板想让停下来加油的次数尽可能的少。幸运的是，卡车的油箱容量无限大。一开始卡车的油箱存有P升油，距离目标小镇L公里远。请计算何老板最少停下来加油几次？

0 <= L <= 1,000,000
1 <= N <= 10,000
1 <= P <= 1,000,000

车能开多远开多远，如果走到x位置油箱空了，意味着之前应该加油，在哪里加油呢？在0到x间没有加过油的油量最多的加油站加油。

每次选出油量最多的加油站，用大根堆即可。

堆的应用1：钓鱼 NKOI 1122

何老板打算钓鱼h小时，他家附近有n个池塘，这些池塘分布在一条直线上，何老板将它们按离家的距离编号，依次为L1，L2，…，Ln，何老板家门口就是第1个池塘，所以他到第1个池塘是不用花时间的。

何老板可以任选多个池塘垂钓，并且在每个池塘他都可以呆上任意长的时间，但呆的时间必须为**5分钟的倍数**（5分钟为一个单位时间），他也可以在任意一个池塘结束今天的钓鱼活动。

已知从池塘Li到池塘Li+1要花去ti个单位时间，每个池塘的上鱼率预先也是已知的，池塘Li在第一个单位时间内能钓到的鱼为Fi，并且每过一个单位时间在单位时间内能钓到的鱼将**减少一个常数di**，现在请你编一个程序计算何老板最多能钓到多少鱼。

$1 \le h \le 16$　$2 \le n \le 25$　$0 \le di \le 100$　$0 \le Fi \le 100$　$0 \le di \le 100$

堆的应用1：钓鱼 NKOI 1122

问题分析：怎么处理来回奔走于各个池塘的情况？

问题答案：不会出现来回奔走的情况，**只会从左往右走**

因为在第i个池塘x分钟钓鱼后，离开一段时间后再次回到i号池塘钓鱼y分钟，一定没有直接在i号池塘钓x+y分钟优秀，因为来回要浪费时间，所以不会出现重复经过同一池塘的情况。

方法一：DP

阶段：从左往右依次讨论以每个池塘作为终点的情况

状态：**f[i][j]**表示在j个单位时间内，在前i个池塘中最多能钓的鱼数

决策：在第i个池塘钓多长时间的鱼

方程：**f[i][j] = max{ f[i-1][j], f[i-1][j-k-walk[i]]+GetFish[i][k] }**

f[i-1][j]表示不在第i个池塘钓鱼

GetFish[i][k]表示在第i个池塘钓k分钟能钓到的鱼数

walk[i]表示从i-1号池塘到达i号池塘行走花费的时间

f[i-1][j-k-walk[i]] 总共j分钟，从i-1到i行走花费walk[i]分钟，i号池塘钓鱼花费k分钟，剩余j-k-walk[i]分钟，这j-k-walk[i]是前i-1个池塘可以使用的钓鱼时间。

时间复杂度：O(n*h*h)

堆的应用1：钓鱼 NKOI 1122

方法二：贪心+堆

由于终点不固定，而且只能从左往右走，所以枚举终点，依次讨论每个点作为终点的情况:

假设以i号池塘作为终点，那么先用总时间减去从1号池塘到i号池塘行走耗费的时间,**Fish_Time=h-walk[i]**,那么剩下的时间(Fish_Time)就是用来钓鱼的时间了。

现在就相当于现在可以在1到i号池塘间任意瞬间移动了，每一个单位时间选当前能够钓到鱼最多的池塘来钓就行了，选最大用大根堆实现。

具体操作，先把1到i号池塘单位时间能够钓鱼的数量存入大根堆：

1.每次从堆顶取出能钓到最大数量鱼的池塘，并在那里钓一个单位时间的鱼，将钓到的鱼累加钓鱼的总数上;

2.把该湖下一个单位时间能钓鱼的数量更新(减去di)后，重新存入堆中；

3.总钓鱼时间减去一个单位的时间，重复该过程，直到没有时间或没有鱼可钓了为止;

时间复杂度：O(n*h*logh)

堆的应用1：钓鱼 NKOI 1122　参考代码

```
struct node {   int fi,di;  }a[30], Now;                  //a[i].fi表示i号池塘单位时间的钓鱼量，a[i].di表示钓鱼量的减少常数

bool operator<(node a,node b)  {  return a.fi<b.fi;  }   //按照单位时间的钓鱼量建立大根堆

priority_queue <node> q;

main()
{
        ……
        for (i=1;i<=n;i++)                                  //依次讨论以每个池塘作为钓鱼的终点
        {
           Sum= 0;
           Time = h - walk[i];                              //h为钓鱼的总时间，walk[i]表示从起点到i号池塘行走耗费的时间
           for ( j=1;j<=i;j++)q.push(a[j]);                 //将前i个池塘按单位时间钓鱼数量存入大根堆
           while (Time>0)                                   //讨论每个单位时间在哪一个池塘钓鱼
           {
                Now = q.top();                              //取出堆顶元素，堆顶代表当前单位时间内能够钓鱼最多的那个池塘
                q.pop();
                Sum = Sum + Now.fi;                        //Max记录下钓鱼总量。
                Now.fi =Now.fi - Now.di;
                if (Now.fi<0) Now.fi = 0;                       //注意：有可能出现单位时间钓鱼数量小于0的情况，应强行置0
                q.push(Now);
                Time--;
           }
           if (Sum>ans) { ans = Sum; }          //更新答案
        }
        ……
}
```

堆的应用2：合并果子 NKOI 1134

有n堆果子，多多决定把所有的果子合成一堆。

每一次合并，多多可以把两堆果子合并到一起，消耗的体力等于两堆果子的重量之和。所有的果子经过n-1次合并之后，就只剩下一堆了。合并果子时总共消耗的体力等于每次合并所耗体力之和。求这个最小的体力耗费值。 n<=10000

huffman树模型：

哈夫曼树一般是二叉树，建树的方法就是每次选择两个权值最小的点，删除这2个点，加入一个权值是这两个点之和的新点进去。并且使这被删除的2个点的父亲成为那个新点。

具体而言，将n堆果子看作n个数字加入到一个小根堆，每次合并相当于连续两次取出堆顶元素，将取出的两个数求和，再次加入到堆中。

堆的应用2：合并果子之 K叉哈夫曼树

有n堆果子，多多决定把所有的果子合成一堆。

每一次合并，多多可以把K堆果子合并到一起，消耗的体力等于K堆果子的重量之和。所有的果子经过若干次合并之后，就只剩下一堆了。合并果子时总共消耗的体力等于每次合并所耗体力之和。 求这个最小的体力耗费值。

K叉huffman树模型：

K叉哈夫曼树是棵K叉树，建树的方法就是每次选择K个权值最小的点，删除这K个点，加入一个权值是这K个点之和的新点进去。并且使这被删除的K个点的父亲成为那个新点。

具体而言，将n堆果子看作n个数字加入到一个小根堆，每次合并相当于连续K次取出堆顶元素，将取出的K个数求和，再次加入到堆中。

然而每次选择k个权值最小的点的时候容易让最后一次合并的时候的点不足k个。假设最初有n个点，最后有1个点，每次合并删除k个点又放进1个点。

那么易得：（n-1）是（k-1）的倍数。如果（n-1）%（k-1）！=0，那么就要再放入（k-1-（n-1）%（k-1））个虚拟点，并且它们的权值为0，它们也参与求最小k个点。

有兴趣的话可以挑战NKOI3425

堆的应用3：奶牛大学 NKOI 2294

题目描述：

有C头奶牛想申请就读"奶牛大学"，奶牛大学是所公益性的学校，所有学员的学费都由学校承担，每头奶牛都在申请书上到标明了自己的高考成绩和所需的学费。今年该校只招收N头奶牛（N为奇数），有金额为F的助学资金可以用于解决新入学奶牛的学费。

校长贝茜希望在**不超过助学资金**的情况下，使得招收的N头奶牛的高考**成绩的中位数尽可能高**，请你求出这个最大可能的中位数。

这里说说奇数个数组成的集合的中位数，例如集合3,8,9,7,5的中位数为7，有两个数小于7，有两个数大于7。

1<=N<=19999　　N<=C<=100000　　0<F<=$2*10^9$

1<=每头牛的高考分数<=$2*10^9$　　1<=每头牛的学费<=100000

样例输入：

```
3 5 70        //分别是N,C,F
30 25         //一头奶牛的CAST分数和它所需的财政学费数a
50 21
20 20
5 18
35 30
```

分数为5,35,50的奶牛，中位数为35．总的资金要求为$18+30+21=69$

样例输出：

```
35
```

补充一下：什么是中位数

中位数（Median）统计学名词，是指将一组数据按大小顺序排列起来，形成一个有序数列，处于数列中间位置的数据就称为中位数，用Me表示。

例如数列a={ 1, 3, 8, 10, 25 }，数字8就是该数列的"中位数"。

当数列的项数N为奇数时，处于中间位置的变量值即为中位数；当N为偶数时，中位数则为处于中间位置的2个变量值的平均数。

例如数列b={ 1, 3, 8, 10, 25, 38 }，则数列b的中位数为(8+10)/2=9

堆的应用3：奶牛大学 NKOI 2294

问题解法：堆

分析过程：

要求选出的N头牛的成绩的中位数尽可能大，我们可以考虑依次讨论每头奶牛的成绩是否适合作为中位数。

一.先按高考分数Score由小到大排序：

显然，排序后，能够作为答案的成绩一定在[n/2+1...C-n/2]这个下标范围之间。

二.若k位于这个范围[n/2+1...C-n/2]，那么Score[k]是否是一个合理的中位数呢？

既然k是选出的中位数的下标，也就意味着:

在[1...k-1]间定要选出n/2头牛，我们希望选总学费尽量少n/2头奶牛，设该学费总额为Left[k]

在[k+1...C]间定也要选出n/2头牛，我们也希望选总学费尽量少n/2头奶牛，设该学费总额为Right[k]

若Left[k]+Right[k]+Money[k]<=F，则Score[k]为一个可行的答案。

最终找出满足条件 Left[k]+Right[k]+Money[k]<=F 的最大的一个k，它对应的Score[k]即为答案。

三.怎样快速求出[n/2+1...C-n/2]区间中每一个数对应的Left[ ]和Right[ ]值呢？

1.建立一个大根堆，把最左边的n/2头牛的学费Money[ ]存到堆中，并记录下堆中奶牛的学费总和Sum。

2.依次讨论n/2+1到C-n/2这段区间的奶牛:

<1>若当前讨论的第k头牛，则Left[k]=Sum。

<2>若Money[k]小于堆顶元素，则用Money[k]替换堆顶元素，调整堆，并且更新它们的总和Sum

<3>继续讨论下一头奶牛(第k+1头)

3.求Right[ ]值与求Left[ ]同理

四.时间复杂度　　排序O(ClogC)+讨论 O(ClogC)

堆的应用3：奶牛大学 NKOI 2294
问题解法：优先队列
核心代码：

1.数据结构

```
struct node
{
  int Score,Money,Left,Right;
} Cow[100005];
```

2.按成绩由小到大排序

```
bool cmp(node a,node b)
{ return a.Score<b.Score; }

  sort(Cow+1,Cow+1+C,cmp);
```

3.按学费建立大根堆

```
priority_queue<int>Q;
for(int i=1;i<=N/2;i++)
{
      q.push(Cow[i].Money);
      Sum+=Cow[i].Money;
}
```

4.求k左边总学费最少的n/2头奶牛的学费值Left

```
 for(k=N/2+1;k<=C-N/2;k++)
 {
                Cow[k].Left=Sum;
                tmp=Q.top();
                if(Cow[k].Money<tmp)
                {
                        Q.pop();
                        Q.push(Cow[k].Money);
                        Sum=Sum-tmp+Cow[k].Money;
                }
 }
```

推荐一道关于中位数的思维题：NKOJ3551 士兵站队

堆的应用4：分配防晒霜 NKOI 1587

有C头奶牛去日光浴，出门前他们要抹防晒霜。第i头奶牛适合的最小和最大的防晒值SPF分别为minSPF_i和maxSPF_i。如果某头奶牛涂的防晒霜的SPF值过小，那么阳光仍然能把她的皮肤灼伤；如果防晒霜的SPF值过大，则会使日光浴与躺在屋里睡觉变得几乎没有差别。

奶牛们准备L瓶防晒霜。第i瓶防晒霜的SPF值为SPF_i。第i瓶防晒霜可供cover_i头奶牛使用。每头奶牛只能涂某一个瓶子里的防晒霜，而不能把若干个瓶里的混合着用。

最多有多少奶牛能在不被灼伤的前提下，享受到日光浴的效果？

1 <= C , L <= 2500
1 <= minSPF_i <= 1,000;　　minSPF_i <= maxSPF_i <= 1,000

堆的应用4：分配防晒霜 NKOI 1587

问题分析：第i瓶防晒霜可供哪些奶牛使用？

我们可以考虑**贪心**的思想：
1.在还没有分配防晒霜的奶牛中，将所有防晒值下限不大于第i瓶防晒霜防晒值(minSPF<=SPF[i])的奶牛都找出来；
2.从这些奶牛中选择防晒值上限不小于第i瓶防晒霜(maxSPF>=SPF[i])且该值尽可能小的奶牛，将第i瓶防晒霜分配给它们使用。
3.实现上述操作，我们容易想到将这些奶牛按防晒值上限用小根堆维护即可。

具体实现：贪心+小根堆

1.将奶牛按防晒值下限(minSPF)由小到大排序，将防晒霜按防晒值(SPF)由小到大排序；

2.依次讨论每瓶防晒霜,设当前讨论第i瓶: 将还没有分配防晒霜的且防晒值下限不大于SPF[i]的奶牛存入小根堆，该堆以奶牛防晒值上限(maxSPF)为关键字；

3.依次取出堆顶元素，若堆顶元素的值(maxSPF)不小于SPF[i]，表示对应的牛可用第i瓶防晒霜，涂该牛后将第i瓶防晒霜可用的次数减一；

4.在第3步完中，若取出的堆顶元素的值(maxSPF)小于SPF[i]，表示该牛无法涂到防晒霜。因为防晒霜是按防晒值由小到大顺序讨论的，当讨论到第i瓶防晒霜时，当前第i瓶也无法涂，后面的防晒霜也无法涂到了，因为后面防晒霜的防晒值都不小于第i瓶。而前面的i-1瓶防晒霜已讨论完毕，前i-1瓶该牛都没有被涂到，

5.回到第2步，继续讨论第i+1瓶防晒霜；

6.时间复杂度**O(ClogC)**

每头奶牛只有一次进堆的机会

堆的应用4：分配防晒霜 NKOI 1587 参考代码

```
//已事先排好序
//k标记当前讨论的奶牛的编号，i标记当前讨论的防晒霜的编号
for(k=1,i=1; i<=L; i++)                                    //总共L瓶防晒霜，依次讨论每一瓶
{
        while (k<=C && cow[k].minSPF<=cream[i].SPF)        //共C头牛，选出防晒值下限不大于当前防晒霜的奶牛
        {
                Q.push(cow[k].maxSPF);
                k++;
        }

        while (!Q.empty() && cream[i].cnt>0)    //!Q.empty()表示还有防晒下限不超过i号防晒霜的奶牛没有分配霜
        {                                       //cream[i].cnt记录i号防晒霜可用的次数
                tmp=Q.top();
                Q.pop();
                if (tmp>=cream[i].SPF)          // 奶牛上限不小于第i瓶，说明这头奶牛可以被涂上i号防晒霜
                {
                        ans++;
                        cream[i].cnt--;
                }
               // else 这头奶牛不能被涂上，因为cream是按SPF排过序的，没有比这瓶更小的SPF了
        }
}
```

堆的应用5：黑盒序列 NKOI 3645

有一个只能存放整数的序列叫“黑盒序列”，一开始序列为空。对于该序列，我们有ADD和GET两种操作：

ADD(x)：表示往序列中添加一个值为x的整数；

GET(y)：表示在第y次ADD操作后，取出序列中第k小的一个数，并将其输出。(k初值为1，每执行一次GET操作后，k的值会加1)

0<=N,M<=30000

样例说明：

操作顺序	k	黑盒序列	输出结果
1 ADD(3)		3	
GET	1	3	3
2 ADD(1)		1, 3	
GET	2	1, 3	3
3 ADD(-4)		-4, 1, 3	
4 ADD(2)		-4, 1, 2, 3	
5 ADD(8)		-4, 1, 2, 3, 8	
6 ADD(-1000)		-1000, -4, 1, 2, 3, 8	
GET	3	-1000, -4, 1, 2, 3, 8	1
GET	4	-1000, -4, 1, 2, 3, 8	2
9 ADD(2)		-1000, -4, 1, 2, 2, 3, 8	

堆的应用5：黑盒序列 NKOI 3645

问题分析：在一个常常更新的数列中，常常去查找第k小数

方法一：用堆暴力枚举

对于每一个GET操作，我们把当前数列的所有数存入一个小根堆，然后用k次取出堆顶元素，其中第k次取出的堆顶元素就是数列的第k小数。

时间复杂度O(n*m*logm)

方法二：1个大根堆 + 1个小根堆

假设当前数列有t个数，我们怎么快速找目前第k小的数呢？

原理很简单：

1.将通过ADD操作新加入数列的数字全部存入大根堆(两次GET操作之间加入的数字)；

2.删除大根堆中最大的t-k+1个数，也就是最后大根堆中只留下了k-1个数字(也就是目前数列的前k-1小个数)；

3.在第2步中，每从大根堆里面删除一个数，就将该数加入到小根堆，最后小根堆里面有了t-k+1个数；

4.第3步完成后，小根堆的堆顶元素一定是当前这t个数中第k小的。

因为前k-1小的数在大根堆里面，于是将小根堆堆顶的数字删除，并将该数字加入到大根堆，大根堆里面就有了k个数了，也就是前k小的数字。

5.当讨论下一个GET操作时，也就是求第k+1小的数字，上一次GET操作前数列中的所有数字已存入了两个堆中，我们回到步骤1继续执行；

堆的应用5：黑盒序列 NKOI 3645 参考代码

```
.....
priority_queue<int,vector<int>,less<int> >Qmax;        //申明了一个大根堆
priority_queue<int,vector<int>,greater<int> >Qmin;     //申明了一个小根堆
.....
Get[0]=0;
for(k=1;k<=m;k++)
{
    for(i=Get[k-1]+1;i<=Get[k];i++)Qmax.push(Add[i]);  //将两次GET间新加入的数字入大根堆
    while(Qmax.size()>k-1)                             //大根堆中只保留目前前k-1小的数字
    {
        Qmin.push(Qmax.top());                         //多的数字从大根堆中删除，并加入到小根堆
        Qmax.pop();
    }
    Qmax.push(Qmin.top());                             //小根堆的堆顶元素一定是当前第k小数
    Qmin.pop();
    printf("%d\n",Qmax.top());
}
.....
```