> ## J题题解报告：贪心+（分块思想？）
>
> 题意：n个纸币和n个商品，不找零钱，k次改变纸币面值操作，求全买下来的最小花费面值。无法全买下来输出"NIE"。

两行杂乱的数据看着很糟心

那就先对这两行数据sort一下，排个序，从大到小排

考虑题目最终的三种情况，

1. k次操作不够用
2. k次操作刚好够买下全部商品
3. 能够买下全部商品后k仍有剩余

首先考虑第一种情况：如何判断？

从大到小依此枚举各个商品，在纸币中查询是否存在能够支付该商品的纸币

如果没有，就花费一次操作

当花费操作大于k时，我们认为，无法把全部商品都买下

第一种情况判断条件找到

接下来考虑第三种情况：在我们已经找到为每个商品匹配最优纸币的前提下

从配对好的商品和纸币中找出差价最大的几组，让纸币的价格降到商品的价格。

~~处处都是贪心的思想。~~

最关键的情况和最需要考虑的一个问题：

k次操作刚好可以买下、如何分配最优纸币和商品配对方案

可以这样考虑：为每个枚举到的商品找到当前不小于商品价格的纸币中的最小的那个纸币（the least upper bound), 在不使用魔法的情况下这个配对方法绝对是最优的。

如果遇到了所有纸币都比当前商品小的情况，就花费一次操作，让某个纸币的价格和该商品相同并支付。

那么问题来了：这个纸币如何选择？

答案是：不用选择！

还记得我们怎么给存在不少于商品价格纸币的商品挑纸币的吗？

我们那使用运用了贪心思想，挑的都是最优的。而所有纸币都比当前商品小的情况下挑选的魔法改变的纸币，就是没有配对，剩下来的那些纸币。

换句话说，就是我们不必要担心选哪个纸币用魔法来改变其面值，因为这并不影响最终结果。

下面来跟着代码一起看看具体怎么实现的

同时解释下为什么我用了分块的思想？（我也不确定算不算）

我采用链式方法存储纸币，因为配对后的纸币不能再筛选，相当于删除操作，直接用数组存时间复杂度会很高。

```
for(int i=1;i<=n;++i){
 pre[i]=i+1;
 nex[i]=i-1;
}//初始条件下，每个纸币的nex纸币和pre纸币
pre[0]=1;
pre[n]=n;
```

同时我还设置了一个尾"指针" tail 用来定位当前纸币最末尾的位置。

当我找到最优配对的纸币时：

```
if(j==tail) tail=nex[tail];
sum+=money[j];
flag[j]=true;//已经用过了
pre[nex[j]]=pre[j];//更新该节点的nex的pre，pre的nex
nex[pre[j]]=nex[j];
diff[++tot]=money[j]-goods[i];//最优配对的差价存起来
```

然后我是这样移动指向纸币的指针的：

```
       //688,666的参数可以随意更改（后者小于前者）
if(j>688) j=nex[j-666];
else j=nex[j];
```

就算我跳到了一个已经用过的点，我也可以不断地往回走，直到遇到了没用过的点。

注意，已经用过的点的pre，nex可能依旧是已经用过的点；但是没用过的点的pre，nex一定时没用过的点。

```
   while(flag[j]) ++j;
   while(money[j]<goods[i]) j=pre[j];
```

配对完之后从差价数组里找最大的几个把剩下的k用掉就行了

```
   sort(diff+1,diff+1+tot);
while(k-->0&&tot){
 sum-=diff[tot--];
}
```

> **时间复杂度O(n*(logn+sqrt(n)):**
>
> 排序就n*logn了，跳跃式查找最优匹配单次大概sqrt（n）