



浙江财经大学

Zhejiang University Of Finance & Economics



贪心算法

信智学院 陈琰宏



教学内容



01

选择不相交区间问题

02

区间选点问题

03

区间覆盖问题

04

流水作业调度问题

第1章 贪心算法



一、贪心算法

贪心算法是从问题的初始状态出发，通过若干次的贪心选择而得出最优值（或较优值）的一种方法。贪心策略并不是从整体上考虑问题，它所做出的选择只是在某种意义上的局部最优解，而许多问题自身的特性决定了该题运用贪心策略可以得到最优解或较优解。



二、贪心算法的特点

1.贪心选择性质：所谓贪心选择性质是指应用同一规划，将原问题变为一个相似的但规模更小的子问题，而后的每一步都是当前看似最佳的选择，这种选择依赖于已做出的选择，但不依赖于未做的选择。

2.最优子结构性质：算法中每一次都取得了最优解(即局部最优解)，要保证最后的结果最优，则必须满足全局最优解包含局部最优解。



三、几个简单的贪心例子

1.最优装载问题:

给 n 个物体, 第 i 个物体重量为 w_i , 选择尽量多的物体, 使得总重量不超过 C 。

【问题分析】

贪心策略: 先装最轻的。



2.部分背包问题：

有 n 个物体，第 i 个物体的重量为 w_i ，价值为 v_i ，在总重量不超过 C 的情况下让总价值尽量高。每一个物体可以只取走一部分，价值和重量按比例计算。

【问题分析】

贪心策略：先拿性价比高的。



3.乘船问题：

有 n 个人，第 i 个人重量为 w_i 。每艘船的载重量均为 C ，最多乘两个人。用最少的船装载所有人。

【问题分析】

贪心策略：最轻的人和最重的人配对。



四、贪心的经典应用

1、选择不相交区间问题：

给定 n 个开区间 (a_i, b_i) ，选择尽量多个区间，使得这些区间两两没有公共点。

【算法】

首先按照结束时间 $b_1 \leq b_2 \leq \dots \leq b_n$ 的顺序排序，依次考虑各个活动，如果没有和已经选择的区间冲突，就选；否则就不选。

【正确性】

假设 $b_j < b_i$ 且 (a_j, b_j) 与 (a_i, b_i) 分别与之之前的活动不冲突，如果当前选 (a_j, b_j) ，若 (a_j, b_j) 与 (a_i, b_i) 不冲突则还可以选择 (a_i, b_i) ，答案加一；若 (a_j, b_j) 与 (a_i, b_i) 冲突，因为 $b_j < b_i$ ，所以 (a_j, b_j) 对以后的影响更小。





[3561] 活动安排

设有 n 个活动的集合 $E = \{1, 2, \dots, n\}$, 其中每个活动都要求使用同一资源。而在同一时间内只有一个活动能使用这一资源。每个活动 i 都有一个要求使用该资源的起始时间 s_i 和一个结束时间 f_i 。且 $s_i < f_i$ 。如果选择了活动 i , 则它在时间区间 $[s_i, f_i)$ 内占用资源。若区间 $[s_i, f_i)$ 与区间 $[s_j, f_j)$ 不相交, 则两活动是相容的。选择出相互兼容的活动组成的最大集合。 $n \leq 100000$ 。

[3561] 活动安排

输入格式：第一行一个整数 n ；接下来的 n 行，每行两个整数 s_i 和 f_i 。

输出格式：输出互相兼容的最大活动个数。

输入样例

4

1 3

4 6

2 5

1 7

输出样例

2

```
1  #include <bits/stdc++.h>
2  using namespace std;
3  struct work{
4      int s; //开始时间
5      int f; //结束时间
6  } a[1005];
7
8  bool cmp(work a, work b) {
9      return a.f < b.f; //按照结束时间排序
10 }
11
12 int n, ans, now;
```



```
14 int main() {
15     scanf("%d", &n);
16     for (int i=1; i<=n; i++)
17         scanf("%d%d", &a[i].s, &a[i].f);
18     sort(a+1, a+1+n, cmp);
19     for (int i=1; i<=n; i++) {
20         if (a[i].s>=a[now].f) {
21             ans++; // 当前活动的开始时间>=上一个活动的结束时间
22             now=i;
23         }
24     }
25     printf("%d", ans);
26     return 0;
27 }
```



2、区间选点问题

给定 n 个闭区间 $[a_i, b_i]$ ，在数轴上选尽量少的点，使得每个区间内都至少有一个点（不同区间内含的点可以是同一个）。

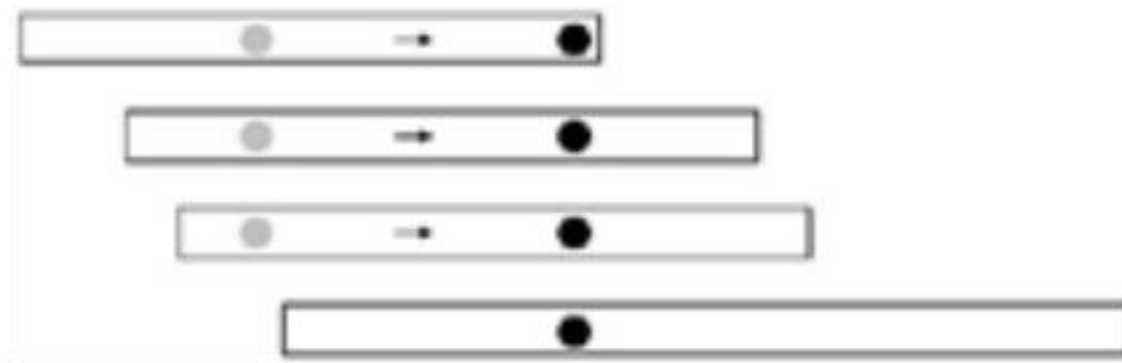


【算法】

首先按照结束位置从小到大排序。然后从区间1到区间n进行选择：对于当前区间，若集合中的数不能覆盖它，则将区间末尾的数加入集合。

贪心策略是：取最后一个。

如下图，如果选灰色点，则移动到黑色点会更优。



[3562] 种树

一条街道的一边有几座房子。因为环保原因居民想要在路边种些树，路边的地区被分割成 n 块，并被编号为 $1 \cdots n$ ，每块大小为一个单位尺寸并最多可种一棵树。每个居民想在门前种些树并指定了三个数 b, e, t 这三个数分别表示该居民想在 b 和 e 之间种最少 t 棵树，当然， $b \leq e, t \leq e - b + 1$ ，允许居民想种树的子区域可以交叉。出于资金紧缺的原因，环保部门请你能够满足所有居民种树要求时所需要种的树的最少数量。

输入第一行为 n ，表示路段数。第二行为 h ，表示建议数。下面 h 行描述一条建议： b, e, t 用一个空格分隔。
 $n \leq 30000, h \leq 5000$

输出只有一个数，为满足所有居民的建议，所需要种树的最少数量。

[3562] 种树

输入样例

9

4

1 4 2

4 6 2

8 9 2

3 5 2

输出样例

5

分析

典型的区间选点问题。每一个房子代表一个要求，这个要求对应着一个左端点和右端点，以及一个要求的树数。它就相当于是给定一个区间，在这个区间内插入要求的点数。求的种数的最小数量即为总点数。

将这个问题转化为一个区间选点问题并得出基本思路。

基本思路：种树要种得少，就要尽量让一棵树处在多个区间内。这样，我们只需要尽量在重叠区间种树即可。而重叠部分一定位于区间尾部。因此，我们可以先按照所有区间的结束位置进行排序，之后依次处理每一个区间，先在一个区间的尾部种满足足够的树，对下一个区间，只需要判断还需要多少棵树就在区间的尾部中多少就可以。



```
1  #include<bits/stdc++.h>
2  using namespace std;
3  struct node{
4      int s;//起始位置
5      int e;
6      int num;//[s,e]间要种树的数量
7  }a[5005];
8
9  int used[30005];// 记录每个位置的状态, 0表示没有种树
10
11 bool cmp(node b,node c){
12     return b.e<c.e;
13 }
14
15 int main(){
16     int n,m,ans=0;
17     cin>>n>>m;
18     for(int i=1;i<=m;i++)
19         cin>>a[i].s>>a[i].e>>a[i].num;
20     sort(a+1,a+1+m,cmp);//完成初始化
```

```
22 for(int i=1;i<=m;i++){//对每个区间按顺序进行处理
23     //1先统计该区间的已有的树的数量
24     int k=0;
25     for(int j=a[i].s;j<=a[i].e;j++)
26         if(used[j])k++;
27     if(k>=a[i].num)continue;//该区间数量已够，不用再种
28     //如果不够开始从后往前种树
29     for(int j=a[i].e;j>=a[i].s;j--){
30         if(!used[j]){
31             used[j]=1;//种树
32             k++;//统计该区间的树的数量
33             ans++;//统计总的种树数量
34             if(k==a[i].num)
35                 break;//树已够，退出
36         }
37     }
38 }
39 cout<<ans;
40 return 0;
41 }
```

3、区间覆盖问题

给 n 个闭区间 $[a_i, b_i]$ ，选择尽量少的区间覆盖一条指定线段 $[s, t]$ 。

【算法】

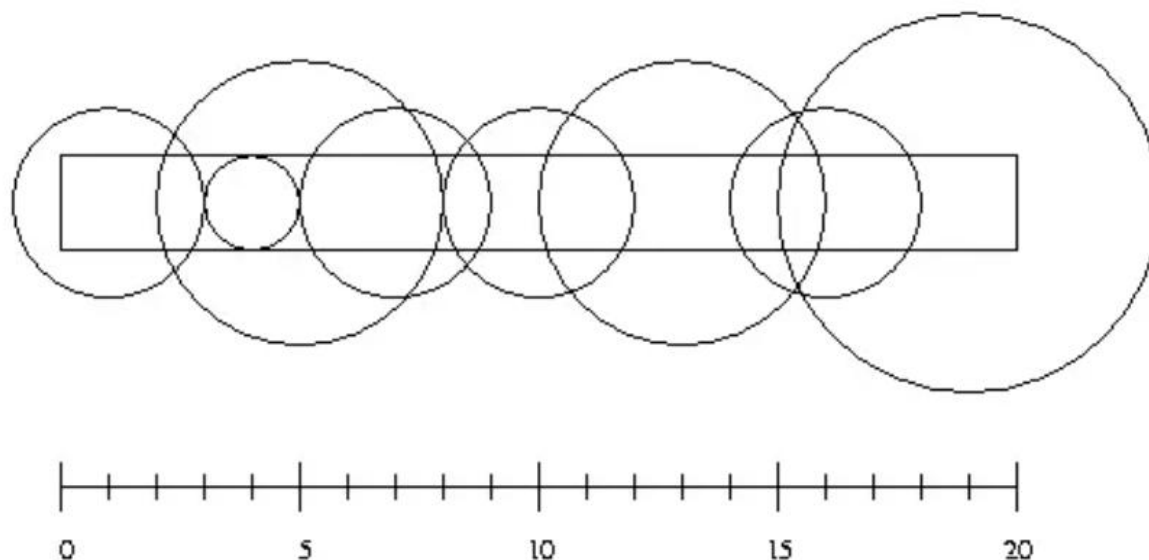
将所有区间按左端点从小到大排序，依次处理每个区间。每次选择覆盖点 s 的区间中右端点坐标最大的一个，并将 s 更新为该区间的右端点坐标，直到选择的区间已包含了 t 为止。

贪心思想：在某个时刻 s ，找一个满足 $a[i] \leq s$ 的 $b[i]$ 最大值即可。



[3599] 喷水装置

长 L 米，宽 W 米的草坪里装有 n 个浇灌喷头，每个喷头都装在草坪的中心线上(离两边各 $W/2$ 米)。我们知道每个喷头的位置(离草坪中心线左端的距离)，以及它能覆盖到的半径。问如果要同时浇灌整块草坪，最少需要打开多少个喷头。若无法覆盖整个草坪则输出-1。



[3599] 喷水装置

输入包含若干组测试数据。第一行一个整数 T 表示数据组数；每组数据的第一行是整数 n 、 L 和 W ； $n \leq 10000$ 接下来的 n 行，每行包含两个整数，给出一个喷头的位
置和浇灌半径（上面的示意图是样例输入第一组数据所描述的情况）。

对每组测试数据输出一个数字，表示要浇灌整块草坪所需喷头数目的最小值。如果所有喷头都打开也不能浇灌整块草坪，则输出 -1 。

输入样例

```
3
8 20 2
5 3
4 1
1 2
7 2
10 2
13 3
16 2
19 4
3 10 1
3 5
9 3
6 1
3 10 1
5 3
1 1
9 1
```

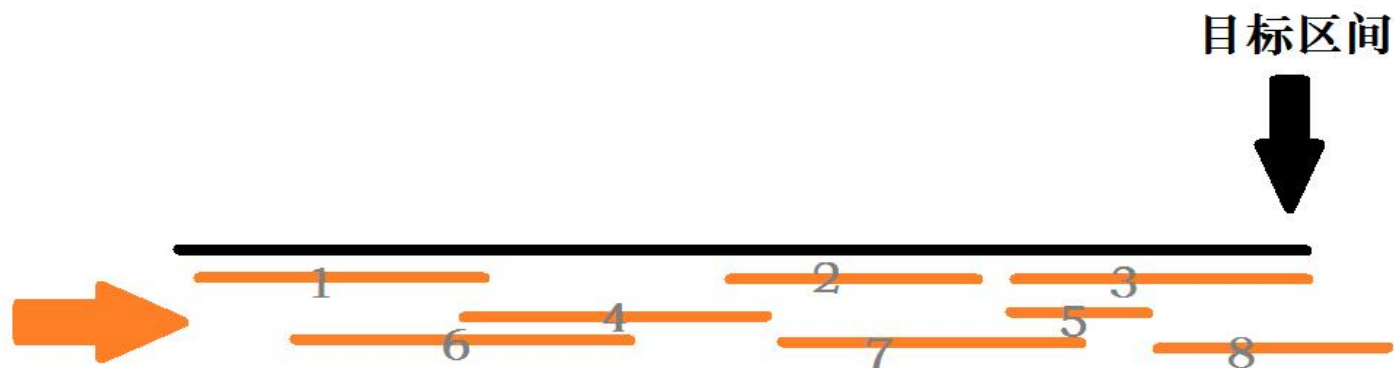
输出样例

```
6
2
-1
```

分析

题目简化：给定一条线段（区间）1（其端点为A, ）和 k 个区间 (p, q) ，请选用最少的区间覆盖这一条线段（区间）
贪心策略

将所有区间按照左端点从小到大排序，选择覆盖点A的区间中右端点最大的一个，并将A更新为这个区间右端点的坐标，再次选择覆盖点A的区间中右端点最大的一个，将A更新为这个区间右端点的坐标，如此往复，直到覆盖了点B
一些小细节。



分析

对于本题，还有一些小细节需要注意，如下：

读入的是喷水装置（即一个圆）的半径 r ，而不是覆盖的范围，覆盖范围应该是：

$$a[k].x = o - \sqrt{r^2 - (w/2)^2}$$

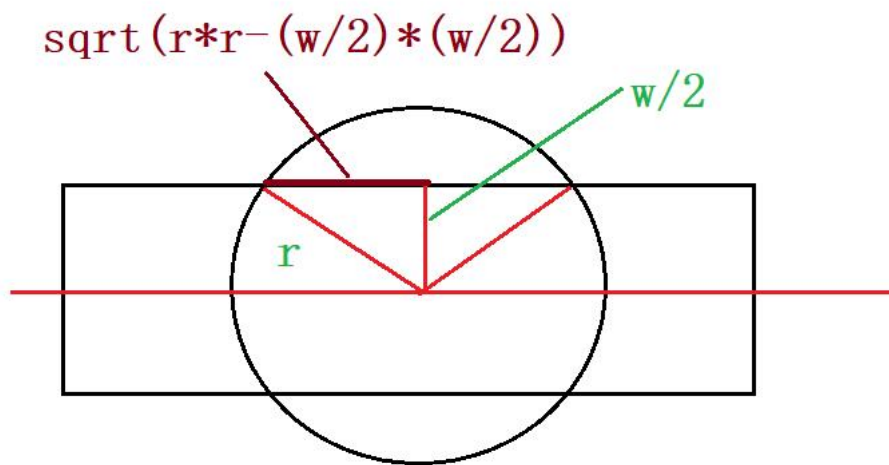
$$a[k].x = o + \sqrt{r^2 - (w/2)^2}$$

这里 x 和 y 是区间的左右端点， o 为喷水装置的坐标，

即喷水装置喷水范围的圆心， r 为其半径， w 为草坪的宽度

如果喷水装置的半径小于草坪的宽度，那么这个喷水装置肯定不能用，

因为它一块区间都覆盖不了



```
3  int T;
4  struct node{
5      double x,y;
6  }a[20005]; //注意数组大小
7  int main()
8  {
9      scanf("%d",&T);
10     while(T-->0)
11     {
12         int n,l,w;
13         scanf("%d%d%d",&n,&l,&w);
14         int cnt = 0;
15         for(int i = 1;i <= n;i++)
16         {
17             int p,q;
18             scanf("%d%d",&p,&q);
19             if(2 * q <= w)continue;
20             cnt++;
21             a[cnt].x = p - sqrt(q * q - w * w/4.0);
22             a[cnt].y = p + sqrt(q * q - w * w/4.0);
23         }
```

```

24 double t = 0;
25 int ans = 0, flag = 0;
26 while(t < 1)
27 { //在所有左端点小于上一个区间右端点的线段中选择右端点最大的一个
28     ans++;
29     double s = t;
30     for(int i = 1; i <= cnt; i++)
31     {
32         if(a[i].x <= s && t < a[i].y) t = a[i].y;
33     }
34     if(t == s && s < 1){
35         //如果找不到两个区域之间断开了或者所有区域长度之和小于规定长度就输出-1
36         printf("-1\n");
37         flag = 1;
38         break;
39     }
40 }
41 if(flag == 0) printf("%d\n", ans);
42 }
43 return 0;

```

```

4  #include<algorithm>
5  using namespace std;
6
7  struct node
8  {
9      double x,y;//x-y这一区间的长度
10 }a[20010];
11
12 int n,last,ans;
13 double st,ed,l,w;
14 int cmp(node n1,node n2)
15 {
16     return n1.x<n2.x;//头越小，整个区间就越长
17 }
18 double gougu(double r)//勾股定理
19 {
20     return sqrt(r*r-(w/2)*(w/2));
21     //利用三角形的勾股定理来求出覆盖的长度
22 }

```

```

23 int main()
24 {
25     int t;
26     scanf("%d",&t);
27     while(t--)
28     {
29         scanf("%d%lf%lf",&n,&l,&w);
30         double o,r;
31         for(int i=1;i<=n;i++)
32         {
33             scanf("%lf%lf",&o,&r);
34             if(r<(w/2)) a[i].x=a[i].y=0;
35             /*
36             如果浇灌的半径大于宽的一半，其实就是长度大于宽的话，
37             那么这个区间就不需要记录，因为这个可以直接实现
38             */
39             else a[i].x=o-gougu(r),a[i].y=o+gougu(r);
40             // 否则利用勾股求出最小的头和最大的尾
41         }
42     }

```

```

43 sort(a+1,a+n+1,cmp);//这是贪心的一步
44 st=0,ed=0,last=0,ans=0;
45 //st是用来保存当前所到的区间的头 //ed是用来保存当前所到区间的尾
46 //last是用来表示当前到的喷头数 //ans是用来保存已经使用的喷头
47 while(ed<l && last<=n)//当还有喷头并且没有到尽头的时候进入
48 {
49     for(int i=last+1;i<=n;i++)
50     {
51         if(a[i].x<=st)//如果当前的区间的头小于之前的,就记录
52         {
53             if(a[i].y>ed)
54             {
55                 ed=a[i].y;
56                 last=i;
57             }//说明找到了最新数据的区间也就是最长的(贪心)
58         }
59         else break;
60     }
61     if(ed==st)
62     //如果这个尾等于头,说明就算打开全部喷头也做不到,就输出-1

```

```

61     if(ed==st)
62         //如果这个尾等于头，说明就算打开全部喷头也做不到，就输出-1
63     {
64         ans=-1;
65         break;//不能return 0是因为有多组数据
66     }
67     st=ed; ans++;
68     /*否则就可以把当前的头记录成尾巴，使用一个喷头，
69     说明这一个区间是成立中的最大的 */
70 }
71
72 if(last>n) printf("-1\n");//如果最后用到的喷头数大于拥有的，输出-1
73 else printf("%d\n",ans);//否则输出用到的最少的喷头数
74 }
75 return 0;
76
77 }

```

4、流水作业调度问题

【例题】 加工生产调度

【问题描述】

某工厂收到了 n 个产品的订单,这 n 个产品分别在A、B两个车间加工,并且必须先先在A车间加工后才可以到B车间加工。

某个产品 i 在A、B两车间加工的时间分别为 a_i 、 b_i 。怎样安排这 n 个产品的加工顺序,才能使总的加工时间最短。这里所说的加工时间是指: 从开始加工第一个产品到最后所有的产品都已在A、B两车间加工完毕的时间。



[1290] 加工生产调度

某工厂收到了 n 个产品的订单，这 n 个产品分别在A、B两个车间加工，并且必须先先在A车间加工后才可以到B车间加工。

某个产品 i 在A、B两车间加工的时间分别为 A_i 、 B_i 。询问怎样安排这 n 个产品的加工顺序，才能使总的加工时间最短。这里所说的加工时间是指：从开始加工第一个产品到最后所有的产品都已在A、B两车间加工完毕的时间。

输入一行仅一个数据 n ($0 < n < 1000$)，表示产品的数量。接下来 n 个数据是表示这 n 个产品在A车间加工各自所要的时间（都是整数）。最后的 n 个数据是表示这 n 个产品在B车间加工各自所要的时间（都是整数）。输出第一行一个数据，表示最少的加工时间。

[1290] 加工生产调度

输入样例

5

3 5 8 7 10

6 2 1 4 9

输出样例

34

【算法分析】

要使时间最短，则就是让机器的空闲时间最短。
B机器在加工过程中，有可能要等待A机器。
最后一个部件在B机器上加工，A机器也在等待B机器的完工。

要使总的空闲时间最少，就要把在A机器上加工时间最短的部件最先加工，这样使得B机器能以最快的速度开始加工；把在B机器上加工时间最短的部件放在最后加工。这样使得A机器能尽快的等待B机器完工。

所以我们要做的事就是：

- （1）就要把在A车间加工时间最短的部件优先加工，这样使得B车间能以最快的速度开始加工
 - （2）把放在B车间加工时间最短的产品放在最后加工，这样使得最后A车间的空闲时间最少
-



于是我们可以设计出这样的贪心法：

设 $M_i = \min\{a_i, b_i\}$

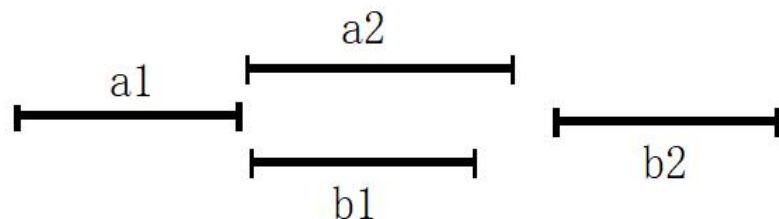
将 M 按照从小到大的顺序排序。然后从第1个开始处理，若 $M_i = a_i$ ，则将它排在从头开始的作业后面，若 $M_i = b_i$ ，则将它排在从尾开始的作业前面。



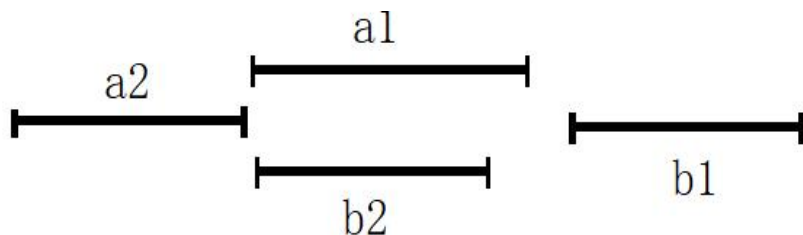
如何确定贪心的顺序

首先，我们假设只有2个产品，在A车间加工的时间为 a_1, a_2 ，在B车间加工的时间为 b_1, b_2 。我们假设先加工产品1的方案较优。

如果先加工1，再加工2，所需时间即为最后加工完2所需的时间。也就是 $a_1 + \max(b_1, a_2) + b_2$ 。



反过来，如果先加工2，再加工1，所需时间为 $a_2 + \max(b_2, a_1) + b_1$ 。



如何确定贪心的顺序

因为我们假设了先加工产品1的方案较优，所以前一种方案的时间更少，也就是 $a1 + \max(b1, a2) + b2 < a2 + \max(b2, a1) + b1$ 。

移项，得到 $\max(b1, a2) - a2 - b1 < \max(b2, a1) - b2 - a1$

然后我们发现不等式两边较大的数都被消掉了，原式即为

$-\min(b1, a2) < -\min(b2, a1)$ 也就是

$$\min(a1, b2) < \min(a2, b1)$$

$$\min(a_i, b_j) < \min(a_j, b_i)$$

可以用贪心思想排序的题都有这么一条性质：如果2个物品按某种方法排序时结果较优，那么多个物品按这种方法排序时结果一定最优。



如何确定贪心的顺序

输入样例

5

3 5 8 7 10

6 2 1 4 9

输出样例

34

设 $M_i = \{a_i, b_i\}$

$(a_1, a_2, a_3, a_4, a_5) = (3, 5, 8, 7, 10)$

$(b_1, b_2, b_3, b_4, b_5) = (6, 2, 1, 4, 9)$



如何确定贪心的顺序

贪心：以样例数据为例：

$(a_1, a_2, a_3, a_4, a_5) = (3, 5, 8, 7, 10)$

$(b_1, b_2, b_3, b_4, b_5) = (6, 2, 1, 4, 9)$

则 $(m_1, m_2, m_3, m_4, m_5) = (3, 2, 1, 4, 9)$ // $m_i = \min(a_i, b_i)$

排序之后为： $(m_3, m_2, m_1, m_4, m_5)$ // m_i 从小到大排序

1 2 3 4 5

处理 m_3 ，因为 $m_3=b_3$ ，所以产品3安排在后面(,,, 3); //从后往前

处理 m_2 ，因为 $m_2=b_2$ ，所以产品2排在后面(,,, 2, 3); //从后往前

处理 m_1 ，因为 $m_1=a_1$ ，所以产品1安排在前面(1,,, 2, 3); //从前往后

处理 m_4 ，因为 $m_4=b_4$ ，所以产品4安排在后面(1,, 4, 2, 3); //从后往前

处理 m_5 ，因为 $m_5=b_5$ ，所以产品5安排在后面(1, 5, 4, 2, 3)。 //从前往后

从而得到加工的顺序 1, 5, 4, 2, 3。计算出最短的加工时间 34。

A 机器:	3	10	7	5	8
B 机器:	6	9	4	2	1
花费时间:	9	22	24	27	34



```
1  #include <iostream>
2  #include <algorithm>
3  using namespace std;
4  struct node{
5      int a,b;//a、b阶段的花费时间
6      int flag;//如果a阶段花费的时间小于b阶段 =0
7  }item[1004];
8
9  bool cmp(node x,node y)
10 {
11     if(x.flag==y.flag)
12     {
13         if(x.flag==0)//ta<tb的按照ta从小到大
14             return x.a<y.a;
15         return x.b>y.b;//ta>=tb的按tb从大到小排序
16     }
17     //ta<tb的订单先做
18     return x.flag<y.flag;
19 }
```

```
21 int main()
22 {
23     int n;
24     scanf("%d",&n);
25     for(int i=0;i<n;i++) scanf("%d",&item[i].a);
26     for(int i=0;i<n;i++) scanf("%d",&item[i].b);
27     for(int i=0;i<n;i++){
28         if(item[i].a<item[i].b)
29             item[i].flag=0;
30         else
31             item[i].flag=1;
32     }
33     sort(item,item+n,cmp);
34     int A=0,B=0;
35     for(int i=0;i<n;i++)
36     {
37         A+=item[i].a;
38         B=max(B,A)+item[i].b; //等待A完成的时间+B的时间
39     }
40     printf("%d\n",B); //最终为所有b完成的时间
41     return 0;
```

5、带限期和罚款的单位时间任务调度

问题：有 n 个任务，每个任务都需要1个时间单位执行，任务 i 的截止时间 d_i ($1 \leq d_i \leq n$)表示要求任务 i 在时间 d_i 结束时必须完成，误时惩罚 w_i 表示若任务 i 未在时间 d_i 结束之前完成，将导致 w_i 的罚款。

确定所有任务的执行顺序，使得惩罚最小。



【贪心方法】

要使罚款最少，我们显然应尽量完成 $w[i]$ 值较大的工作。

此时，我们可以将工作按 $w[i]$ 从大到小进行排序，然后按照排好的顺序依次对工作进行安排，安排的规则为：使处理工作 i 的时间既在 $d[i]$ 之内，又尽量靠后；如果 $d[i]$ 之内的时间都已经排满，就放弃处理此项工作。



[3466] 智力大冲浪

小伟报名参加中央电视台的智力大冲浪节目。本次挑战赛吸引了众多参赛者，主持人为了表彰大家的勇气，先奖励每个参赛者 m 元 ($m < \text{maxlongint}$)。先不要太高兴！因为这些钱还不一定都是你的？！接下来主持人宣布了比赛规则：

首先，比赛时间分为 n 个时段 ($n \leq 500$)，他又给出了很多小游戏，每个小游戏都必须在规定期限 t_i 前完成 ($1 \leq t_i \leq n$)。如果一个游戏没能在规定期限前完成，则要从奖励费 m 元中扣去一部分钱 w_i ， w_i 为自然数，不同的游戏扣去的钱是不一样的。当然，每个游戏本身都很简单，保证每个参赛者都能在一个时段内完成，而且都必须从整时段开始。主持人只是想考考每个参赛者如何安排组织自己做游戏的顺序。作为参赛者，小伟很想赢得冠军，当然更想赢取最多的钱！

注意：比赛绝对不会让参赛者赔钱！

输入共4行：第1行为m，表示一开始奖励给每位参赛者的钱；第2行为n，表示有n个小游戏；第3行有n个数，分别表示游戏1到n的规定完成期限；第4行有n个数，分别表示游戏1到n不能在规定期限前完成的扣款数。

输出表示小伟能赢取最多的钱。

输入样例

10000

7

4 2 4 3 1 4 6

70 60 50 40 30 20

10

输出样例

9950

样例解释：

在时刻4 完成第一个任务

在时刻2 完成第二个任务

在时刻3 完成第三个任务

在时刻1 完成第四个任务

第五个任务没法完成扣30

第六个任务没法完成扣20

在时刻5或6完成任务6

思路：

因为不同的小游戏不能准时完成时具有不同的扣款权数，而且是求问题的最优解，所以很容易想到用贪心算法求解。

贪心策略是让扣款数额大的尽量在规定的期限内完成。

这样我们就先把这些任务按照扣款的数额进行排序，把大的排在前面，进行放置。

假如罚款最多的一个任务的完成期限是 k ，我们应该把它安排在哪个时段完成呢？应该放在小于等于 k 的最靠后的时间段。一旦出现一个不可能在规定期限内完成的任务，就直接扣钱



```
1  #include<bits/stdc++.h>
2  using namespace std;
3  struct yx{ //构造一个结构体，将游戏的规定期限t与扣款数q关联
4      int t;
5      long q;
6  }a[505]; //生成储存游戏信息的a数组
7  bool cmp(yx a,yx b){ //按扣款数的大小降序排列
8      return a.q>b.q;
9  }
10 int main(){
11     int m,n;
12     cin>>m>>n; //输入初始金额m，以及游戏数n
13     int c[505]={0}; //用于表示每一阶段是否已被占用，0表示未占用
14     for(int i=1;i<=n;i++){ //输入规定期限
15         cin>>a[i].t;
16     }
17     for(int i=1;i<=n;i++){ //输入扣款金额
18         cin>>a[i].q;
19     }
```

```
20 | sort(a+1,a+n+1,cmp);
21 | //将a数组按扣款数的大小降序排列，便于下一步为扣款数高的游戏
22 | //优先安排时间
23 | for(int i=1;i<=n;i++){
24 |     //此循环用于给每个游戏寻找完成阶段
25 |     for(int j=a[i].t;j>=1;j--){
26 |         //从游戏规定期限开始由后向前寻找空余时间
27 |         if(c[j]==0){ //若能在期限内能找到未占用阶段
28 |             c[j]=1; //则将该阶段标为1，即已占用，并终止寻找
29 |             break;
30 |         }
31 |         if(j==1)m-=a[i].q;
32 |         //若反推至第一阶段仍未寻找到未占用阶段，则扣除相应罚款
33 |     }
34 | }
35 | cout<<m; //输出剩余的金额
36 | return 0;
37 | }
```

[3563] 数列极差

在黑板上写了N个正整数组成的一个数列，进行如下操作：每次擦去其中的两个数a和b，然后在数列中加入一个数 $a \times b + 1$ ，如此下去直至黑板上剩下一个数，在所有按这种操作方式最后得到的数中，最大的为max，最小的为min，则该数列的极差定义为 $M = \max - \min$ 。请你编程，对于给定的数列，计算极差。

输入第一行为一个数N表示正整数序列长度（ $0 \leq N \leq 10$ ）第二行为N个正整数。所有值包括中间运算产生的值，都在整型类型范围内。

输出文件只有一行，为相应的极差d

输入样例

3

1 2 3

输出样例

2

分析

我们可以先举个便于观察的例子：

现有3个正整数组成的数列，元素分别为 a, b, c ，我们规定 $a < b < c$ 。

于是这个数列便有3种操作方式（在这里我称一次操作为合并）：

1. 先合并 a, b ，再与 c 合并。

$a \quad b \quad c$

$ab+1 \quad c$

$abc+c+1$

结果为 $abc+c+1$ 。

2. 先合并 a, c ，再与 b 合并。

$a \quad b \quad c$

$ac+1 \quad b$

$abc+b+1$

结果为 $abc+b+1$ 。

3. 先合并 b, c ，再与 a 合并。

$a \quad b \quad c$

$bc+1 \quad a$

$abc+a+1$

结果为 $abc+a+1$



分析

我们可以先举个便于观察的例子：

现有3个正整数组成的数列，元素分别为 a, b, c ，我们规定 $a < b < c$ 。
于是这个数列便有3种操作方式（在这里我称一次操作为合并）：

1. 先合并 a, b ，再与 c 合并。

a	b	c
ab+1		
abc+1		

2. 先合并 a, c ，再与 b 合并。

a	b	c
ac+1	b	
abc+b+1		

3. 先合并 b, c ，再与 a 合并。

a	b	c
	bc+1	
abc+a+1		



我们可以发现，三种方式最后的结果都有含有相同的数 $abc+1$ ，唯一不同的仅是三种方式分别加的是 c, b, a 。

我们还可以发现，先合并 a, b （即两个较小的数）得到的答案最大，先合并 b, c （即两个较大的数）得到的答案最小。

大胆假设：计算 $\max$ 时可以一直合并数列中两个最小的数，计算 $\min$ 时可以一直合并数列中两个最大的数，直至无法合并。

贪心求解：

求 $\max$ ：每次选取数列中最小的两个相乘

求 $\min$ ：每次选取数列中最大的两个相乘



```
1  #include<cstdio>
2  #include<algorithm>
3  using namespace std;
4  bool cmpmin(int n1,int n2)
5  { //min是小, 函数就是从小到大排序
6      return n1<n2;
7  }
8  bool cmpmax(int n1,int n2)
9  { //max是大, 函数就是从大到小排序
10     return n1>n2;
11 }
12 int a[51000]; //原始输入数组
13 int b[51000]; //复制数组
```

```
14 int main()
15 {
16     int n;
17     scanf("%d",&n);
18     for(int i=1;i<=n;i++)
19     {
20         scanf("%d",&a[i]);
21         b[i]=a[i]; //复制
22     }
23     sort(a+1,a+n+1,cmpmin); //a数组从小到大排序
24     sort(b+1,b+n+1,cmpmax); //b数组从大到小排序
25     for(int i=1;i<n;i++)
26     {
27         a[i+1]=a[i]*a[i+1]+1;
28         b[i+1]=b[i]*b[i+1]+1;
29         sort(a+i+1,a+n+1,cmpmin);
30         sort(b+i+1,b+n+1,cmpmax);
31     }
32     printf("%d\n",a[n]-b[n]);
33     return 0;
34 }
```

[3590] 数列分段

对于给定的一个长度为N的正整数数列 $A[i]$ ，现要将其分成连续的若干段，并且每段和不超过M（可以等于M），问最少能将其分成多少段使得满足要求。

输入文件的第1行包含两个正整数N，M，表示了数列 $A[i]$ 的长度与每段和的最大值，第2行包含N个空格隔开的非负整数 $A[i]$ ，如题目所述。

输出格式

输出文件仅包含一个正整数，输出最少划分的段数。

输入样例

```
5 6
4 2 4 5 1
```

输出样例

```
3
```

ans来累加组数，用s和c来统计和只要没有超过限制，我们就继续加。如果超过限制，那么组数就要增加，并且c为超过的这项。注：要把ans置为1，因为最少也有一组

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  int a[100001],n,m,i,c=0,ans=1,s=0;
4  int main()
5  {
6      cin>>n>>m;
7      for(i=1;i<=n;i++)cin>>a[i];
8      for(i=1;i<=n;i++){
9          s=c+a[i];
10         if(s<=m)c+=a[i];
11         else{//超过了就重新计数
12             ans++;
13             c=a[i];
14         }
15     }
16     cout<<ans;
17 }
```

[3591] 线段

在一个数轴上有 n 条线段，现要选取其中 k 条线段使得这 k 条线段两两没有重合部分（端点可以重合），问最大的 k 为多少。

输入文件的第1行为一个正整数 n ，下面 n 行每行2个数字 a_i, b_i ，描述每条线段。

输出文件仅包括1个整数，为 k 的最大值

输入样例

3

0 2

2 4

1 3

输出样例

2

分析

因为题目要
可以保证为后面
贪心选可行的最

```
1  #include<
2  #include<
3  using nam
4  struct se
5      int B
6      int E
7  };
8  segment s
9  bool cmp(
10 { // 按照右
11     return
12 }
```

```
13 int main(){
14     int n,ans=0;
15     cin>>n;
16     for(int i=0;i<n;i++)
17         cin>>seg[i].B>>seg[i].E;
18     sort(seg,seg+n,cmp);
19     int end=0;
20     for(int i=0;i<n;i++){
21         if(seg[i].B>=end){
22             // 当前线段左端点>=上一条右端点
23             ans++; // 类似会议安排的选择
24             end=seg[i].E;
25         }
26     }
27     cout<<ans;
28     return 0;
29 }
```

[3592] 家庭作业

老师在开学第一天就把所有作业都布置了，每个作业如果在规定的时间内交上来的话才有学分。每个作业的截止日期和学分可能是不同的。例如如果一个作业学分为10，要求在6天内交，那么要想拿到这10学分，就必须在第6天结束前交。

每个作业的完成时间都是只有一天。例如，假设有7次作业的学分和完成时间如下：

作业号	1	2	3	4	5	6	8
期限	1	1	3	3	2	2	6
学分	6	7	2	1	4	5	1

该例子最多可以获得15学分，其中一个完成作业的次序为2，6，3，1，7，5，4，注意可能还有其它方法。你的任务就是找到一个完成作业的顺序获得最大学分。对于所有数据， $N \leq 1000000$ ，作业的完成期限均小于700000。

分析

这题与智力大冲浪类似，先按照学分从大到小排序。对于每一个作业，按照期限的最大日期来时从大到小扫描（假设作为 a_i 的期限为5天，从5开始5/4/3/2/1 扫描每一天），如果当前的这天没有安排作业就安排掉（即 $vis[i]=0$ ）。如果所有符合要求的位置（做当前作业的时间）都被使用，当前作业的学分就丢弃。

因为这一题数据量很大，完全采 $O(n^2)$ 的时间复杂度会超时，所以需要进行剪枝。

剪枝1： $a[i].t$ 表示作业 i 的时间期限，如果 $\leq a[i].t$ 的位置都被使用，则该作业可以直接放弃。

剪枝2：作业 i 的位置已经找到，则后续的位置不需要在找。



```
1  #include <iostream>
2  #include <cstdio>
3  #include <algorithm>
4  #define ll long long
5  using namespace std;
6  const int MAXN = 1e6 + 5;
7
8  ll n, ans, last;
9  bool vis[MAXN]; //判断当前期限的学分是否获取过
10 struct homework {
11     ll t; //t示作业的完成期限,
12     ll s; //s表示该作业的学分。
13     ll id; //
14 } a[MAXN];
15
16 //按照学分排序 学分大的先处理
17 bool cmp(homework a, homework b) {
18     return a.s > b.s;
19 }
```

```
21 □ int main() {
22     scanf("%lld", &n);
23     for (ll i = 1; i <= n; i++)
24         scanf("%lld%lld", &a[i].t, &a[i].s);
25     sort(a + 1, a + n + 1, cmp);
26 □ for (ll i = 1; i <= n; i++) {
27         if (a[i].t <= last) //last表示<=last的位置都已经被使用
28             continue; //优化, 当前作业的能使用的位置都已经被占用
29         bool ok = false;
30 □ for (ll j = a[i].t; j >= 1; j--) {
31 □     if (!vis[j]) { //如果该时间点 j 没有被用
32         vis[j] = true; // 记录该位置被用
33         ans += a[i].s;
34         ok = true; //找到了一个位置
35         break;
36     }
37     } //ok=false表明<=a[i].t的位置都不可用
38     if (!ok)
39         last = a[i].t; //记录
40 }
41 printf("%lld\n", ans);
42 return 0;
```

[3593] 钓鱼

在一条水平路边，有 n ($2 \leq n \leq 100$) 个钓鱼湖，从左到右编号为1、2、3、 $\dots$ 、 n 。佳佳有 H ($1 \leq H \leq 20$) 个小时的空余时间，他希望用这些时间钓到尽量多的鱼。他从湖1出发，向右走，有选择的在一些湖边停留一定的时间钓鱼，最后在某一个湖边结束钓鱼。佳佳测出从第 i 个湖到第 $i+1$ 个湖需要走 $5 \times T_i$ 分钟的路，还测出在第 i 个湖边停留，第一个5分钟可以钓到鱼 F_i ，以后再每钓5分钟鱼，鱼量减少 D_i 。为了简化问题，佳佳假定没有其他人钓鱼，也不会有其他因素影响他钓到期望数量的鱼。

请设计一个算法求出佳佳能钓到最多鱼的方案。
输入的第一行为湖泊的数量 n ；第二行为钓鱼的可用时间数，以小时记；第三行为 n 个整数，表示第 i 个湖泊第一个5分钟的钓鱼数量；第四行为 n 个整数，表示第 i 个湖泊再钓鱼5分钟鱼量减小值 p ；第五行为 $n-1$ 个整数，表示从第 i 个湖泊到第 $i+1$ 个湖泊所需要的时间数 $5 \times T_i$ 分钟

[3593] 钓鱼

输出最大的钓鱼数量，每个输出占一行

输入样例

```
3           //3个池塘
1           //钓鱼的总时间 (*60)
4 5 6       //第i个湖泊第一个5分钟的钓鱼数量
1 2 1       //第i个湖泊再钓鱼5分钟的鱼量减小值p
1 2         //从第i个湖泊到第i+1个湖泊所需的时间
```

输出样例

35

在第 1 个湖钓 15 分钟，共钓得 $4+3+2=9$ 条鱼；

在第 2 个湖钓 10 分钟，共钓得 $5+3=8$ 条鱼；

在第 3 个湖钓 20 分钟，共钓得 $6+5+4+3=18$ 条鱼；

从第 1 个湖到第 2 个湖，从第 2 个湖到第 3 个湖，共用时间 15 分钟，共得 35 条鱼，并且这是最多的数量。

分析

从1出发，顺序往后面走，故可以枚举它最后停留在了哪个位置，如果最终停留的位置确定了，那么在路上钓鱼的时间确定了，对于剩下钓鱼时间的选择就可以用贪心来做。

即然最后不知道停在哪个湖，那就分类讨论。把停在每个湖的最优解全部求出，再最后求最优解。当我们知道主人公最后停在哪个湖后，她的路径也就唯一确定了（例如佳佳最后停在了第 i 个湖，那么她的路径一定是 $1 \rightarrow 2 \rightarrow 3 \rightarrow \dots \rightarrow i$ ），同时她的纯钓鱼时间可由总空闲时间减去行程时间唯一确定。考虑从哪个湖钓鱼一个5分钟，就相当于在路径 $1 \rightarrow 2 \rightarrow 3 \rightarrow \dots \rightarrow i$ 中的一个节点上“堆”上一个标记表示在这个湖又钓了5分钟的鱼，显然这里可用贪心策略，每次标记目前为止五分钟钓鱼数目最大的那个湖，并使当前记录答案的 $sum_i +=$ 在那个湖又钓的鱼数。



```
2  #include<bits/stdc++.h>
3  using namespace std;
4  const int N=1e2+10;
5  int fish[N],d[N],vis[N],s[N];
6  int n,h,k,ans;
7  bool cmp(int a,int b)
8  {
9      return a>b;
10 }
11 int main()
12 {
13     cin>>n>>h;
14     for(int i=1;i<=n;i++) cin>>fish[i];
15     for(int i=1;i<=n;i++) cin>>d[i];
16     for(int i=2;i<=n;i++) cin>>vis[i];
17     vis[1]=0;
18     h*=12;//这里*12相当于60/5, 也就是把5分钟看做一个单位
```

```
20 //s数组存储从1~i每个湖所有合法时刻的钓鱼数
21 //每次选择是在 s数组中选一个最大的。
22 for(int i=1;i<=n;i++){
23     h-=vis[i];//去掉所有的走路时间花费的时间
24     if(h<=0) break;//剩余时间<=0表明后面的湖走不到
25
26     for(int j=fish[i],t=1;j>0&&t<=h;j-=d[i],t++)
27         s[k++]=j;// 求出当前鱼塘每个合法时刻能钓的鱼的数量
28
29     sort(s,s+k,cmp);
30     int x=0;
31     for(int u=1,v=0;v<k&&u<=h;v++,u++)
32         x+=s[v];//从大到小在时间 h 内能钓鱼的数量
33     if(x>ans) ans=x;
34 }
35 cout<<ans<<endl;
36 return 0;
37 }
```

考虑在第 k 个人之后断开，则环化成链有：

$$\begin{array}{ll} A_{k+1} & S_{k+1} - S_k \\ \dots & \dots \\ A_n & S_n - S_k \\ A_1 & S_1 + S_n - S_k \\ A_2 & S_2 + S_n - S_k \\ \dots & \dots \\ A_k & S_k + S_n - S_k \end{array}$$

又易知 $S_n = \sum c_j - n \times T/m = 0$ ，故求得最小步数为：

$$\sum_{i=1}^n |S_i - S_k| \quad \text{其中 } S_i \text{ 为 } A_i \text{ 的前缀和，即 } S_i = \sum_{j=1}^i A_j$$

该绝对值不等式最小值的求解，就同上一题“货仓选址”异曲同工了

因此 S_k 的选择，就取 S_i 排序后的中位数即可



[6372] 货仓选址

在一条数轴上有 N 家商店，它们的坐标分别为 A_1-A_n 。
现在需要在数轴上建立一家货仓，每天清晨，从货仓到每家商店都要运送一车商品。

为了提高效率，求把货仓建在何处，可以使得货仓到每家商店的距离之和最小。

第一行输入整数 N 。第二行 N 个整数 $A_1 \sim A_N$ 。

输出一个整数，表示距离之和的最小值。

$1 \leq N \leq 100000, \quad 0 \leq A_i \leq 40000$

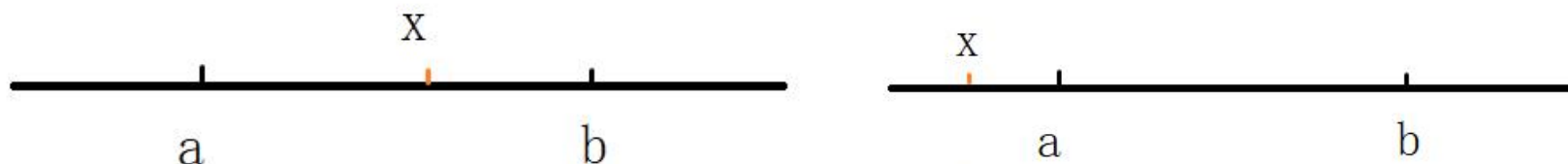
输入样例

4

6 2 9 1

输出样例

12



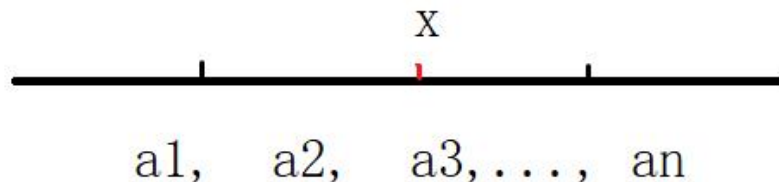
题目要求的是 n 和商店与货仓的距离之和 s 最小，那么我们可以先来看看当商店只有两个的情况下如何选择才能使得 s 最小？

假设区间 $[a, b]$ 的距离是 n ，那么如果货仓 x 选在区间 $[a, b]$ 的任一位置的话， $s=n$ ；

如果选在区间 $[a, b]$ 之外的话， $s>n$ 。

如此可以得到绝对值不等式：

$$|x-a| + |x-b| \geq |a-b|$$



$s = (|a_1 - x| + |a_n - x|) + (|a_2 - x| + |a_{n-1} - x|) + \dots + (|a_{n/2} - x| + |a_{n/2+1} - x|)$, 如果n是偶数, 中间两项一组, 如果n是奇数, 中间一项单独一组。根据绝对值不等式, **如果x取到中位数(等号成立)**, 我们可以推导出:

$$\begin{aligned}
 & (|a_1 - x| + |a_n - x|) + (|a_2 - x| + |a_{n-1} - x|) + \dots + (|a_{n/2} - x| + |a_{n/2+1} - x|) \geq \\
 & |a_n - a_1| + |a_{n-1} - a_2| + \dots +
 \end{aligned}$$

货仓应该选在最中间的两个点之间, 此时的s一定是最小的。当商店个数是奇数时, 中位数只有一个; 当个数是偶数时, 中位数有两个, 由于货仓可以选择的是闭区间之间的任何一点, 因此可以统一取成 $(n+1)/2$ 下取整, 下标从1开始。

算法分析

- 1、把 $A[0] \sim A[N-1]$ 排序，设货仓在X坐标处。
- 2、仓库应该建在中位数处，把A进行排序，

当N为奇数时，货仓建在 $A[(N - 1)/2]$ 处，

当N为偶数时，仓库建在 $A[(N - 1)/2 + 1]$ 处。

时间复杂度($O(n \log(n))$)



算法分析

```
1  #include <bits/stdc++.h>
2  using namespace std;
3
4  const int N = 100005;
5  int n, res;
6  int a[N];
7  int main()
8  {
9      scanf("%d", &n);
10     for (int i = 0; i < n; i ++ )
11         scanf("%d", &a[i]);
12     sort(a, a + n);
13     for (int i = 0; i < n; i ++ )
14         res += abs(a[i] - a[n >> 1]);
15     printf("%d\n", res);
16     return 0;
17 }
```



[3006] 糖果传递

有 n 个小朋友坐成一圈，每人有 a_i 颗糖果。每人只能给左右两人传递糖果。每人每次传递一颗糖果的代价为 1。求使所有人获得均等糖果的最小代价。

输入第一行有一个整数 n ，表示小朋友个数；

在接下来 n 行中，每行一个整数 a_i 。

$n \leq 10^6$ ，保证答案可以用 64 位有符号整数存储。

输出格式

输出使所有人获得均等糖果的最小代价。

输入样例

4

1

2

5

4

输出样例

4

输入样例

4

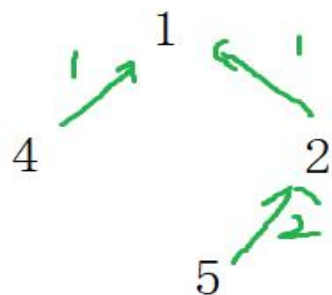
1 2 5 4

输出样例

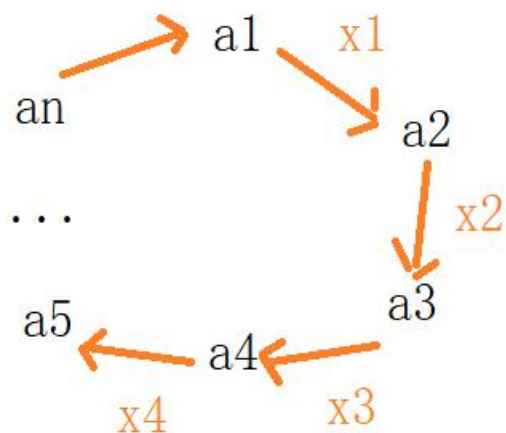
4

总共4个人，12颗糖，最终每人3颗糖。

$$1+1+2=4$$



分析



设一共有 M 个人，每个人初始的纸牌数量为 a_i ，纸牌总数为 T 。显然当 $M \nmid T$ 时，方案不存在，现只考虑方案存在的情况。

设 a_1 向 a_2 传递 x_1 张牌（正数是给，负数是拿）

， a_2 向 a_3 传递 x_2 张牌， a_3 向 a_4 传递 x_3 张牌， $\dots$ ， a_{n-1} 向 a_n 传递 x_n 张牌。

则交换次数为：

$$\sum \text{abs}(x_i)$$

该问题可以理解为“环形均分纸牌”问题。

环形区间问题，第一想到的就是 **破环成链**。

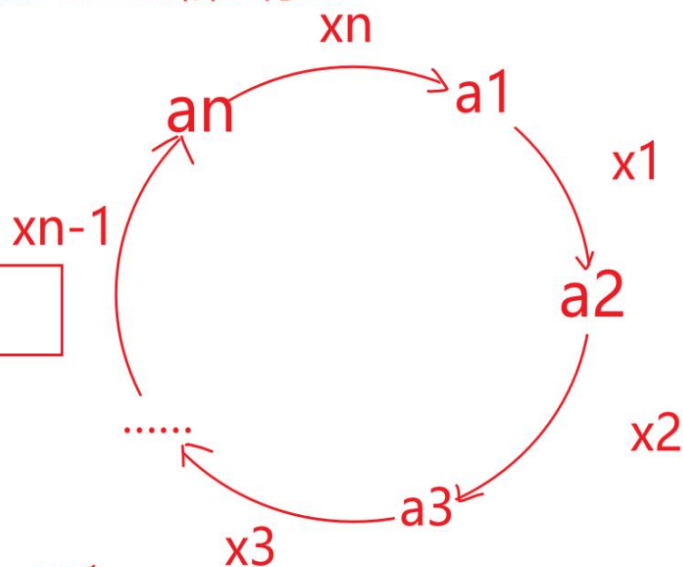
可以直接枚举断点的位置，然后做一遍线性纸牌均分，但是时间复杂度为 $O(n^2)$ ，会超时，考虑数学方法推导。

1 假设有第1小朋友有 a_1 颗糖，给第2个小朋友 x_1 颗糖，从n获得 x_n ，此时他有 $a_1 - x_1 + x_n$ 颗糖，同理第2个有 $a_2 - x_2 + x_1$ ，第3有.....

2 每个小朋友的目标为平均数 avg ,
列出约束方程为

$$\begin{aligned} a_1 - x_1 + x_n &= \text{avg} \\ a_2 - x_2 + x_1 &= \text{avg} \\ a_3 - x_3 + x_2 &= \text{avg} \\ &\dots\dots \\ a_n - x_n + x_{(n-1)} &= \text{avg} \end{aligned}$$

目标为
 $\min(|x_1| + |x_2| + \dots + |x_n|)$



方程中的等式关系，用只含 x_n 的式子整体代换目标的 $x_1 \dots x_{n-1}$

$$\begin{aligned}x_1 &= a_1 - \text{avg} - x_n \\x_2 &= a_2 + x_1 - \text{avg} = (a_1 + a_2) - 2 * \text{avg} - x_n \\x_3 &= a_3 + x_2 - \text{avg} = (a_1 + a_2 + a_3) - 3 * \text{avg} - x_n \\&\dots\dots \\x_{(n-1)} &= (a_1 + \dots\dots + a_{n-1}) - (n-1) * \text{avg} - x_n\end{aligned}$$

将常数看成 C_i

$$\begin{aligned}x_1 &= c_1 - x_n \\x_2 &= c_2 - x_n \\&\dots\dots\dots \\x_{(n-1)} &= c_{(n-1)} - x_n\end{aligned}$$

$$\min(|c_1 - x_n| + \dots + |c_{n-1} - x_n|)$$

分析

此时这题就和“仓库选址”做法一样，只需要求中位数和其他数的差值的总和就可以了。

```
4  const int N=1000010;
5  typedef long long ll;
6  ll s[N],a[N],sum,avg,c[N],n;
7  ll res=0;
8
9  int main()
10 {
11     cin>>n;
12     for(int i=1;i<=n;i++)
13     {
14         cin>>a[i];
15         sum+=a[i];//
16     }
17     avg=sum/n;//平均值
18     for(int i=1;i<=n;i++)
19     {
20         c[i]=c[i-1]+a[i]-avg;//求出系数
21     }
22     sort(c+1,c+n+1);
23     for(int i=1;i<=n;i++)
24     {
25         res+=abs(c[i]-c[(n+1)/2]);
26     }
27     cout<<res;
28 }
```



[6373] 耍杂技的牛

农民约翰的 N 头奶牛（编号为 $1..N$ ）计划逃跑并加入马戏团，为此它们决定练习表演杂技。

奶牛们不是非常有创意，只提出了一个杂技表演：

叠罗汉，表演时，奶牛们站在彼此的身上，形成一个高高的垂直堆叠。

奶牛们正在试图找到自己在在这个堆叠中应该所处的位置顺序。

这 N 头奶牛中的每一头都有着自己的重量 W_i 以及自己的强壮程度 S_i 。

一头牛支撑不住的可能性取决于它头上所有牛的总重量（不包括它自己）减去它的身体强壮程度的值，现在称该数值为风险值，风险值越大，这只牛撑不住的可能性越高。

您的任务是确定奶牛的排序，使得所有奶牛的风险值中的最大值尽可能的小。

[6373] 耍杂技的牛

输出一个整数，表示最大风险值的最小可能值。
数据范围

$$1 \leq N \leq 50000,$$
$$1 \leq W_i \leq 10,000, 1 \leq S_i \leq 1,000,000,000$$

输入

3

10 3

2 5

3 3



我们先分析每头牛的危险值 = 他前面牛的 w (重量值)和 - 自身的 s (强壮值), 要使每头牛的危险值最小, 这显然是与 w 和 s 同时相关。

首先我们用数学语言描述这道题: (我们记从上到下牛的编号为 $1 \sim n$), 那么我们要求的就是找出一种牛的排列方式, 令 $\max(w_1 + \dots + w_{i-1} - s_i)$ 最小, 记这个值为 val 。为了求排序的方案, 可以交换 $i, i+1$ 牛的位置, 看看满足什么等价条件, 就可以使得交换之后 val 不比之前大。



下面求交换之后val. 不比之前大的等价条件:

(注意到交换 $i, i+1$ 牛的位置不会影响其他牛的风险度, 故只需考察这两头牛。)

交换前:

牛 i :

$$w_1 + \cdots + w_{i-1} - s_i \quad ①$$

牛 $i+1$:

$$w_1 + \cdots + w_{i-1} + w_i - s_{i+1} \quad ②$$

交换后:

牛 i :

$$w_1 + \cdots + w_{i-1} + w_{i+1} - s_i \quad ③$$

牛 $i+1$:

$$w_1 + \cdots + w_{i-1} - s_{i+1} \quad ④$$

val不比之前大, 即 $\max(①, ②) \geq \max(③, ④)$



[2355] 星星还是树（一维扩二维）

在二维平面上有 n 个点，第 i 个点的坐标为 (x_i, y_i) 。请你找出一个点，使得该点到这 n 个点的距离之和最小。该点可以选择在平面中的任意位置，甚至与这 n 个点的位置重合。

输入第一行包含一个整数 n 。接下来 n 行，每行包含两个整数 x_i, y_i ，表示其中一个点的位置坐标。

输出最小距离和，答案四舍五入取整。

数据范围 $1 \leq n \leq 100$ $0 \leq x_i, y_i \leq 10000$





[6980] Yough的最大化

Yough现在有n个物品的重量和价值分别是Ci和Vi，你能帮他从中选出k个物品使得单位重量的价值最大吗？

即求r的最大值

$$r = \frac{\sum v[j]}{\sum c[j]} (j = i_1, i_2, i_3 \dots i_k)$$

输入每组测试数据第一行有两个数n和k，接下来一行有n个数Ci和Vi。(1<=k<=n<=10000) (1<=ci, Vi<=1000000)

输出使得单位价值的最大值。（保留两位小数）

输入样例

3 2

2 2

3 2

2 1

输出样例

0.80

[6980] Youghth的最大化

问题描述：有N个物体，它们的价值用 $v[i]$ 表示，重量用 $c[i]$ 表示。现在要在这N个物体中选取K个物体，使得选出来的这K个物体的总价值除以中重量达到最大值。即取得最大值。

问题转化：构造一个 $x[N]$ 的数组，表示每个数取或不取的状态，显然每一个 $x[i]$ 只有两个取值：0和1，其中1表示取，0表示不取。则整个式子也就可以变成目标式：

$$r = \frac{\sum v[i] * x[i]}{\sum c[j] * x[i]} (i = 1, 2, 3 \dots N)$$

值得注意的是：上式中的 r 是每一组 x 对应求得的当前答案，而我们的目的就是要找到一组 x 使得求出来的 r 得到最大值 R ！（这很重要！）

$$\frac{a_1}{b_1} > \frac{a_2}{b_2} > \frac{a_3}{b_3}$$

这个等式是否成立?

$$\frac{a_1 + a_2}{b_1 + b_2} > \frac{a_1 + a_3}{b_1 + b_3} \quad ?$$



$$r = \frac{\sum v[i] * x[i]}{\sum c[j] * x[i]} (i = 1, 2, 3 \dots N)$$

进一步对式子进行处理：

简单的一项处理：

$$0 = \sum v[i] * x[i] - r * \sum c[i] * x[i]$$

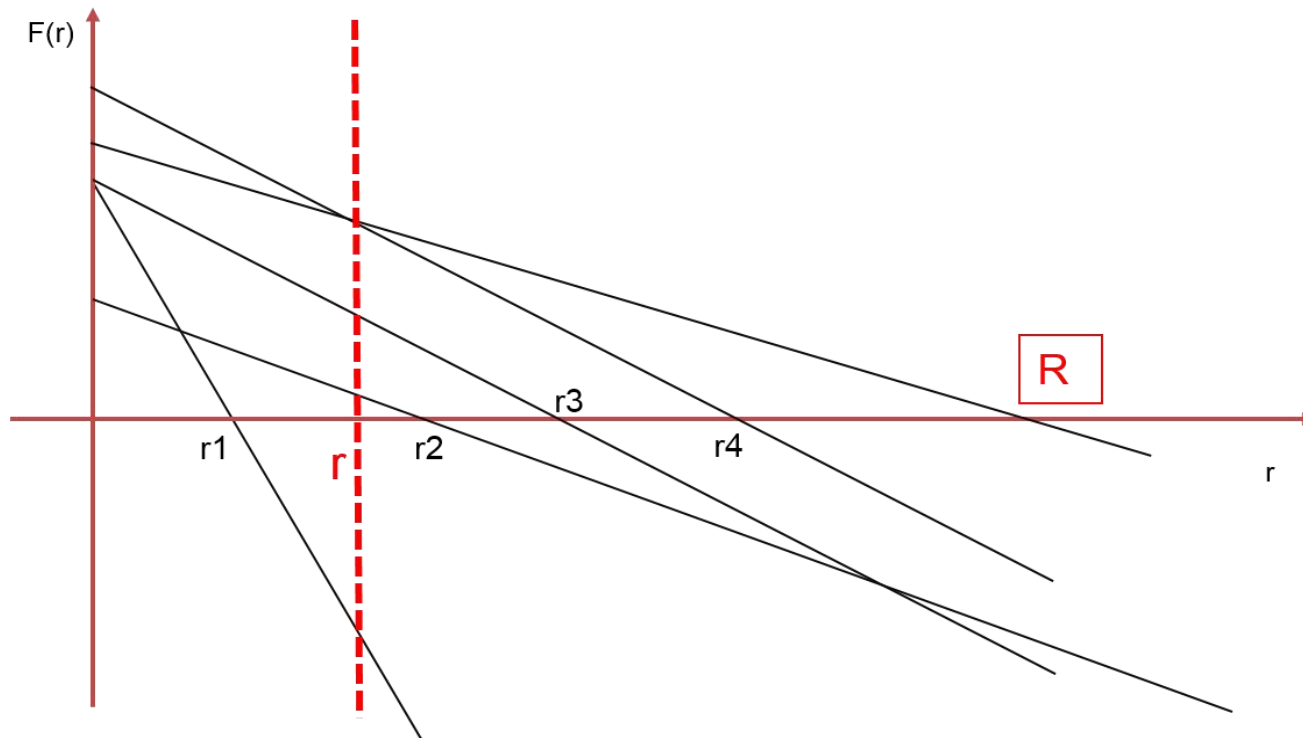
构造一个函数：

$$F(r) = \sum v[i] * x[i] - r * \sum c[i] * x[i]$$

其中F(r)在平面坐标系上体现为一条直线，每一组x[i]都分别唯一地对应一条直线，这些直线的截距均大于等于0、斜率均小于等于0。而这些直线在x轴上的截距就是这一组x求出来的r，而截距的最大值就是我们要求的R。

其中F(r)在平面坐标系上体现为一条直线，每一组x[i]都分别唯一地对应一条直线，这些直线的截距均大于等于0、斜率均小于等于0。而这些直线在x轴上的截距就是这一组x求出来的r，而截距的最大值就是我们要求的R。（如下图所示）

题解: <https://www.cnblogs.com/KirisameMarisa/p/4187637.html>



在x轴上面任取一个 r ，如果至少有一条直线的 $F(r) > 0$ ，那么说明了什么呢？

说明至少还有一条直线与x轴的交点在它的右边，那么这个 r 一定不是最大值，真正的最大值在它的右边。反之，如果所有的 $F(r)$ 都小于0，那么真正的最大值在它的左边

那么前面的结论就可以换种说法，因为我只需判断最大的那个 $F(r)$ 的正负性就行了：

随便取一个 r ，如果 $F(r)_{\max} > 0$ ，则结果 $R > r$ ，反之若 $F(r)_{\max} < 0$ ，则结果 $R < r$ 。直到找到使 $F(r)_{\max} = 0$ 的 r ，那个 r 就是我们要找的结果 R

显然这是一个令人感到兴奋的结论，因为我们自然而然就想到了一种方法：二分法！

今天的课程结束啦.....



下课了...
同学们再见!

