第六章 贪心算法

贪心算法也叫作贪婪算法，是指在求解问题时总做出在当前看来最好的选择，就是不从整体考虑问题，仅在某种意义上的局部最优解。虽然不是所有问题都能得到最优解，但是面对范围广泛的许多问题时，能产生整体最优解或者是整体最优解的近似值。

贪心算法思路：从问题的某一个初始解出发，逐步逼近给定的目标，以便更快的求出更好的解。当达到算法中某一步不能再向前时，就停止算法，给出一个近似解。

缺点：（1）不能保证最后的解是最优解；

（2）不能用来求最大、最小解问题；

（3）只能求满足某些约束条件的可行解范围。

【例1】矩阵选数问题

在N行M列的正整数矩阵中，要求从每行中选出1个数，使得选出的总共N个数的和最大。

【算法分析】

要使总和最大，则每个数要尽可能大，自然应该选每行中最大的那个数。因此，我们设计出如下算法:

```
读入N, M,矩阵数据;
Total = 0;
for (int i = 1;  i <= N; ++ i)
{                                          //对N行进行选择
   选择第i行最大的数,记为K;
   Total += K;
}
输出最大总和Total;
```

从上例中我们可以看出，和递推法相仿，贪心法也是从问题的某一个初始解出发，向给定的目标递推。但不同的是，推进的每一步不是依据某一固定的递推式，而是做一个局部的最优选择，即贪心选择（在例中，这种贪心选择表现为选择一行中的最大整数），这样，不断的将问题归纳为若干相似的子问题，最终产生出一个全局最优解。

特别注意的是，局部贪心的选择是否可以得出全局最优是能否采用贪心法的关键所在。对于能否使用贪心策略，应从理论上予以证明。下面我们看看另一个问题。

【例2】部分背包问题(物品可分割问题)

给定一个最大载重量为M的卡车和S种食品，有食盐，白糖，大米等。已知第i种食品的最多拥有Wi公斤，其商品价值为Vi元/公斤，编程确定一个装货方案，使得装入卡车中的所有物品总价值最大。

输入：
第一行输入一个正整数n（1<=n<=5）,表示有n组测试数据；
随后有n测试数据，每组测试数据的第一行有两个正整数s，m（1<=s<=10）;s表示有s个物品。接下来的s行每行有两个正整数v，w。
输出：
输出每组测试数据中背包内的物品的价值和，每次输出占一行。
样例输入
1
3 15
5 10
2 8
3 9
样例输出
65

【算法分析】

因为每一个物品都可以分割成单位块，单位块的利益越大显然总收益越大，所以它局部最优满足全局最优，可以用贪心法解答，方法如下：先将单位块收益按从大到小进行排列，然后用循环从单位块收益最大的取起，直到不能取为止便得到了最优解。

因此我们非常容易设计出如下算法：

```
问题初始化;                                    //读入数据
按Vi从大到小将商品排序;
 i=1;ans=0;
 for(i=0;i<s;i++)
 {
   if (wi>=m)
        {将重量m第i种商品装入卡车;break; }  //如果卡车满载则跳出循环
   else
        {将wi  重量的物品i装入卡车; m=m-w[i];}
 };
```

在解决上述问题的过程中，首先根据题设条件，找到了贪心选择标准(Vi)，并依据这个标准直接逐步去求最优解，这种解题策略被称为贪心法。

因此，利用贪心策略解题，需要解决两个问题：

首先，确定问题是否能用贪心策略求解；一般来说，适用于贪心策略求解的问题具有以下特点：

①可通过局部的贪心选择来达到问题的全局最优解。运用贪心策略解题，一般来说需要一步步的进行多次的贪心选择。在经过一次贪心选择之后，原问题将变成一个相似的，但规模更小的问题，而后的每一步都是当前看似最佳的选择，且每一个选择都仅做一次。

②原问题的最优解包含子问题的最优解，即问题具有最优子结构的性质。在背包问题中，第一次选择单位重量价值最大的货物，它是第一个子问题的最优解，第二次选择剩下的货物中单位重量价值最大的货物，同样是第二个子问题的最优解，依次类推。

③其次，如何选择一个贪心标准？正确的贪心标准可以得到问题的最优解，在确定采用贪心策略解决问题时，不能随意的判断贪心标准是否正确，尤其不要被表面上看似正确的贪心标准所迷惑。在得出贪心标准之后应给予严格的数学证明。

思考：0-1背包问题

给定一个最大载重量为M的卡车和N种动物。已知第i种动物的重量为Wi，其最大价值为Vi，设定M，Wi，Vi均为整数，编程确定一个装货方案，使得装入卡车中的所有动物总价值最大。

【分析】对于n种动物，要么被装，要么不装，也就是说在满足卡车载重的条件下，如何选择动物，使得动物价值最大的问题。

即确定一组$x1$，$x2$，…，xn，$xi \in \{0,1\}$

$f(x) = max(\sum xi * vi)$ 其中，$\sum(xi * wi) \leq w$

从直观上来看，我们可以按照上例一样选择那些价值大，而重量轻的动物。也就是可以按价值质量比（vi/wi）的大小来进行选择。可以看出，每做一次选择，都是从剩下的动物中选择那些vi/wi最大的，这种局部最优的选择是否能满足全局最优呢？我们来看看一个简单的例子：

设n=3，卡车最大载重量是100，三种动物a、b、c的重量分别是40，50，70，其对应的总价值分别是80、100、150。

情况a：按照上述思路，三种动物的vi/wi分别为2,2,2.14。显然，我们首先选择动物c，得到价值150，然后任意选择a或b，由于卡车最大载重为100，因此卡车不能装载其他动物。

情况b：不按上述约束条件，直接选择a和b。可以得到价值80+100=180，卡车装载的重量为40+50=90。没有超过卡车的实际载重，因此也是一种可行解，显然，这种解比上一种解要优化。

问题出现在什么地方呢？我们看看图23

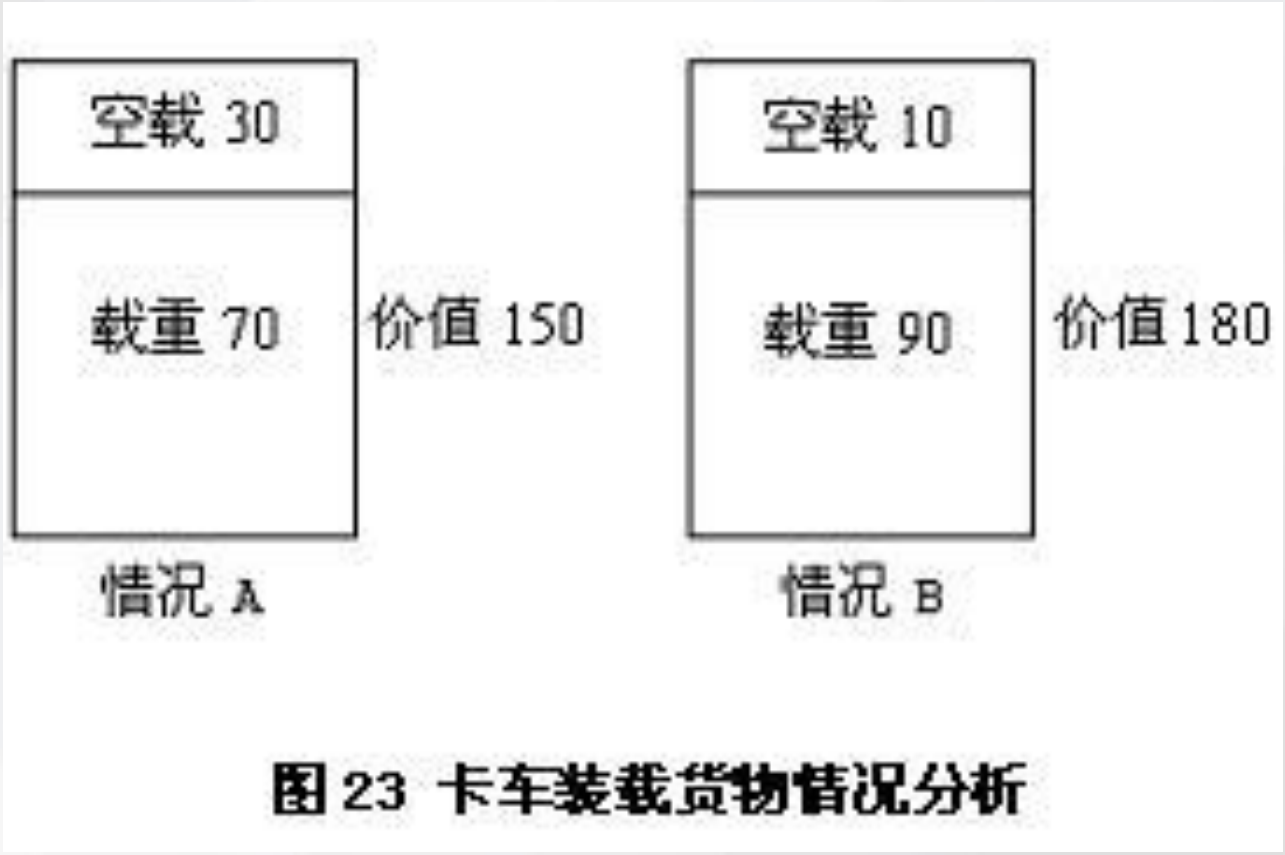


图 23 卡车装载货物情况分析

从图23中明显可以看出，情况a，卡车的空载率比情况b高。也就是说，上面的分析，只考虑了货物的价值质量比，而没有考虑到卡车的运营效率，因此，局部的最优化，不能导致全局的最优化。

因此，贪心不能简单进行，而需要全面的考虑，最后得到证明。

【例3】排队打水问题

有N个人排队到R个水龙头去打水，他们装满水桶的时间为T1，T2，…，Tn为整数且各不相等，应如何安排他们的打水顺序才能使他们花费的时间最少？

【算法分析】

由于排队时，越靠前面的计算的次数越多，显然越小的排在越前面得出的结果越小（可以用数学方法简单证明，这里就不再赘述），所以这道题可以用贪心法解答，基本步骤：

(1)将输入的时间按从小到大排序；

(2)将排序后的时间按顺序依次放入每个水龙头的队列中；

(3)统计，输出答案。

【样例输入】

```
4 2                 //4人打水，2个水龙头
2 6 4 5             //每个打水时间
```

【样例输出】

```
23                  //总共花费时间
```

参考程序主要框架如下：

```
1.  cin>>n>>r;
2.  memset(s,0,sizeof(s));                    //初始化
3.  j=0; minx=0;
4.  for (i=1;i<=n;++i)                        //用贪心法求解
5.   {
6.     j++;
7.     if (j==r+1) j=1;
          //前r个人为一组，第r+1个人回到第一个水龙
8.     s[j]+=a[i];                            //加上等待时间
9.     minx+=s[j];
10.   }
11.  cout<<minx;                              //输出解答
```

【例4】均分纸牌（NOIP2002）

有 N 堆纸牌，编号分别为 1，2，…, N。每堆上有若干张，但纸牌总数必为 N 的倍数。可以在任一堆上取若干张纸牌，然后移动。

移牌规则为：在编号为 1 堆上取的纸牌，只能移到编号为 2 的堆上；在编号为 N 的堆上取的纸牌，只能移到编号为 N-1 的堆上；其他堆上取的纸牌，可以移到相邻左边或右边的堆上。

现在要求找出一种移动方法，用最少的移动次数使每堆上纸牌数都一样多。例如 N=4，4 堆纸牌数分别为： ① 9 ② 8 ③ 17 ④ 6

移动3次可达到目的：

从 ③ 取4张牌放到④（9 8 13 10）->从③取3张牌放到 ②（9 11 10 10）-> 从②取1张牌放到①（10 10 10 10）。

【输入格式】

N（N 堆纸牌，1 <= N <= 100）

A1 A2 … An （N 堆纸牌，每堆纸牌初始数，l<= Ai <=10000）

【输出格式】

所有堆均达到相等时的最少移动次数。

【样例输入】Playcard.in

```
4
9 8 17 6
```

【样例输出】Playcard.out

```
3
```

【算法分析】

如果你想到把每堆牌的张数减去平均张数，题目就变成移动正数，加到负数中，使大家都变成0，那就意味着成功了一半！拿例题来说，平均张数为10,原张数9，8，17，6，变为-1，-2，7，-4，其中没有为0的数，我们从左边出发：要使第1堆的牌数-1变为0，只须将-1张牌移到它的右边（第2堆）-2中；结果是-1变为0，-2变为-3，各堆牌张数变为0，-3，7，-4；同理：要使第2堆变为0，只需将-3移到它的右边（第3堆）中去，各堆牌张数变为0，0，4，-4；要使第3堆变为0，只需将第3堆中的4移到它的右边（第4堆）中去，结果为0，0，0，0，完成任务。每移动1次牌，步数加1。也许你要问，负数张牌怎么移，不违反题意吗？其实从第i堆移动-m张牌到第i+1堆，等价于从第i+1堆移动m张牌到第i堆，步数是一样的。

如果张数中本来就有为0的，怎么办呢？如0，-1，-5，6，还是从左算起（从右算起也完全一样），第1堆是0，无需移牌，余下与上相同；再比如-1，-2，3，10，-4，-6，从左算起，第1次移动的结果为0，-3，3，10，-4，-6；第2次移动的结果为0，0，0，10，-4，-6，现在第3堆已经变为0了，可节省1步，余下继续。

参考程序主要框架如下：

```
cin>>n;
ave=0;step=0;
for (i=1;i<=n;++i)
 {
     cin>>a[i]; ave+=a[i];          //读入各堆牌张数，求总张数ave
 }
ave/=n;                             //求牌的平均张数ave
for (i=1;i<=n;++i) a[i]-=ave;       //每堆牌的张数减去平均数
i=1;j=n;
while (a[i]==0&&i<n) ++i;           //过滤左边的0
while (a[j]==0&&j>1) --j;           //过滤右边的0
while (i<j)
 {
   a[i+1]+=a[i];                    //将第i堆牌移到第i+1堆中去
   a[i]=0;                          //第i堆牌移走后变为0
   ++step;                          //移牌步数计数
   ++i;                             //对下一堆牌进行循环操作
   while (a[i]==0&&i<j) ++i;        //过滤移牌过程中产生的0
 }
cout<<step<<endl;
```

点评：基本题，本题有3点比较关键：一是善于将每堆牌数减去平均数，简化了问题；二是要过滤掉0（不是所有的0，如-2，3，0，-1中的0是不能过滤的）；三是负数张牌也可以移动，这是关键中的关键。

【例5】删数问题（NOI94）

输入一个高精度的正整数N，去掉其中任意S个数字后剩下的数字按原左右次序组成一个新的正整数。编程对给定的N和S，寻找一种方案使得剩下的数字组成的新数最小。输出新的正整数。（N不超过240位）输入数据均不需判错。

【输入】

n

s

【输出】

最后剩下的最小数。

【样例输入】

175438

4

【样例输出】

13

【算法分析】

由于正整数n的有效数位为240位，所以很自然地采用字符串类型存贮n。那么如何决定哪s位被删除呢？是不是最大的s个数字呢？显然不是，大家很容易举出一些反例。为了尽可能逼近目标，我们选取的贪心策略为：每一步总是选择一个使剩下的数最小的数字删去，即按高位到低位的顺序搜索，若各位数字递增，则删除最后一个数字；否则删除第一个递减区间的首字符，这样删一位便形成了一个新数字串。然后回到串首，按上述规则再删下一个数字。重复以上过程s次为止，剩下的数字串便是问题的解了。

例如：n=175438
s=4

删数的过程如下：

n=175438 //删掉7
15438 //删掉5
1438 //删掉4
138 //删掉8
13 //解为13

这样，删数问题就与如何寻找递减区间首字符这样一个简单的问题对应起来。不过还要注意一个细节性的问题，就是可能会出现字符串串首有若干0的情况，甚至整个字符串都是0的情况。按以上贪心策略编制的程序框架如下

```
输入n，s；
 for (i=1;i<=s;++i) {                  //一共要删除s个字符
  for ( j=0;j<len-1;++j )              //从串首开始找，len是n的长度
   if ( n[j]>n[j+1] ) {                //找到第一个符合条件的
    for ( k=j;k<len-1;++k )            //删除字符串n的第j个字符 ， 后面字符往前整
      n[k]=n[k+1];
    break;
   }
  --len;                               //长度减1
 }
 输出n;                                //删去串首可能产生的无用零
```

【例6】拦截导弹问题（NOIP1999）

某国为了防御敌国的导弹袭击，开发出一种导弹拦截系统，但是这种拦截系统有一个缺陷：虽然它的第一发炮弹能够到达任意的高度，但是以后每一发炮弹都不能高于前一发的高度。某天，雷达捕捉到敌国的导弹来袭，由于该系统还在试用阶段。所以一套系统有可能不能拦截所有的导弹。

输入导弹依次飞来的高度（雷达给出的高度不大于30000的正整数）。计算要拦截所有导弹最小需要配备多少套这种导弹拦截系统。

【输入格式】

n颗依次飞来的高度（1≤n≤1000）.

【输出格式】

要拦截所有导弹最小配备的系统数k。

【输入样例】missile.in

389 207 155 300 299 170 158 65

【输出样例】missile.out

2

【输入输出样例】

输入：导弹高度： 7 9 6 8 5

输出：导弹拦截系统K=2

输入：导弹高度： 4 3 2

输出：导弹拦截系统K=1

【算法分析】

按照题意，被一套系统拦截的所有导弹中，最后一枚导弹的高度最低。设：

k为当前配备的系统数；

l[k]为被第k套系统拦截的最后一枚导弹的高度，简称系统k的最低高度（1≤k≤n）。

我们首先设导弹1被系统1所拦截（k←1,l[k]←导弹1的高度）。然后依次分析导弹2，…，导弹n的高度。

若导弹i的高度高于所有系统的最低高度，则断定导弹i不能被这些系统所拦截，应增设一套系统来拦截导弹I(k←k+1,l[k]←导弹i的高度)；若导弹i低于某些系统的最低高度，那么导弹i均可被这些系统所拦截。究竟选择哪个系统拦截可使得配备的系统数最少，我们不妨采用贪心策略，选择其中最低高度最小（即导弹i的高度与系统最低高度最接近）的一套系统p(l[p]=min{l[j]|l[j]>导弹i的高度}；l[p]←导弹i的高度）（i≤j≤k）。这样可使得一套系统拦截的导弹数尽可能增多。

依次类推，直至分析了n枚导弹的高度为止。此时得出的k便为应配备的最少系统数。

参考程序主要框架如下：

```
   k=1;l[k]=导弹1的高度;
  for (i=2;i<=n;++i)
    {     p=0;
          for (j=1;j<=k;++j)
             if (l[j]>=导弹i的高度) {  if (p==0) p=j;
                          else if (l[j]<l[p]) p=j;}    //贪心
             if (p==0) { ++k;l[k]=导弹i的高度; }       //增加一套新系统
                          else l[p]=导弹i的高度;      //贪心,更新第p套系统的最低高度
    }
```

输出应配备的最少系统数K。

参考代码

```
#include <stdio.h>
int main()
{
    int a[1005],k;
    int h,i,j;
    int minI,first;
    freopen("missile.in","r",stdin);
    freopen("missile.out","w",stdout);
    scanf("%d",&h); //第1枚导弹的高度
    k=0;
    a[k]=h;
    while(scanf("%d",&h)==1)
        //while(cin>>h)
    {
        first=1;
        for(i=0;i<=k;i++)
//扫描现有的系统，寻找看是否有可用的系统
        {
            if(a[i]>=h)    //可用使用a[i]来拦截
            {
                if(first==1) {minI=i;first=0;}
                //首次发现可用的系统
                else if(a[i]<a[minI]) minI=i;
            }
        }
        if(first==0)
            a[minI]=h;//使用现有的系统来拦截
        else
        {
            k++;
            a[k]=h;    //新增加一套系统来拦截
        }
    }
    printf("%d\n",k+1);
    return 0;
}
```

【例7】活动选择

学校在最近几天有n个活动，这些活动都需要使用学校的大礼堂，在同一时间，礼堂只能被一个活动使。由于有些活动时间上有冲突，学校办公室人员只好让一些活动放弃使用礼堂而使用其他教室。

现在给出n个活动使用礼堂的起始时间begini和结束时间endi(begini < endi)，请你帮助办公室人员安排一些活动来使用礼堂，要求安排的活动尽量多。

【输入】第一行一个整数n(n<=1000)；接下来的n行，每行两个整数，第一个begini，第二个是endi(begini<endi≤32767)

【输出】 输出最多能安排的活动个数。

【样例输入】

```
11
3 5
1 4
12 14
8 12
0 6
8 11
6 10
5 7
3 8
5 9
2 13
```

【样例输出】

```
4
```

【算法分析】

- 算法模型：给n个开区间（begini,endi），选择尽量多的区间，使得两两不交。
- 做法：首先按照end1≤end2 ≤ … ≤ endn的顺序排序，依次考虑各个活动，如果没有和已经选择的活动冲突，就选；否则就不选。
- 正确性：如果不选end1，假设第一个选择的是endi,则如果endi和end1不交叉则多选一个end1更划算；如果交叉则把endi换成end1不影响后续选择。

【参考程序】

```
#include<iostream>
using namespace std;
int n,begin[1001],end[1001];
void init()
{
  cin>>n;
  for(int i=1;i<=n;i++)
    cin>>begin[i]>>end[i];
}
void qsort(int x,int y)
{
  int i,j,mid,t;
  i=x;j=y;mid=end[(x+y)/2];
  while(i<=j)
    {
      while(end[i]<mid) ++i;
      while(end[j]>mid) --j;
      if(i<=j)
       {
         t=end[j];end[j]=end[i];end[i]=t;

     t=begin[j];begin[j]=begin[i];begin[
     i]=t;
         ++i;j;
       }
    }
  if(x<j) qsort(x,j);
  if(i<y) qsort(i,y);
}
```

```
void solve()
{
  int ans=0;
  for(int i=1,t=-1;i<=n;++i)          //在初始化循环变量的同时，初始化t。
                                      //令t=-1可以使第一个区间与其他区间的操作相同。
   if(begin[i]>=t) {++ans;t=end[i];}  //如果当前活动与之前最后结束的活动
                                      //不冲突，就接受当前活动。
  cout<<ans<<endl;
}
int main()
{
  init();
  qsort(1,n);
  solve();
  return 0;
}
```

【例8】整数区间

请编程完成以下任务：

1.从文件中读取闭区间的个数及它们的描述；

2.找到一个含元素个数最少的集合,使得对于每一个区间,都至少有一个整数属于该集合，输出该集合的元素个数。

【输入】

首行包括区间的数目$n, 1 \leq n \leq 10000$,接下来的n行,每行包括两个整数a,b,被一空格隔开,$0 \leq a \leq b \leq 10000$,它们是某一个区间的开始值和结束值。

【输出】

第一行集合元素的个数,对于每一个区间都至少有一个整数属于该区间,且集合所包含元素数目最少。

【样例输入】

```
4
3 6
2 4
0 2
4 7
```

【样例输出】

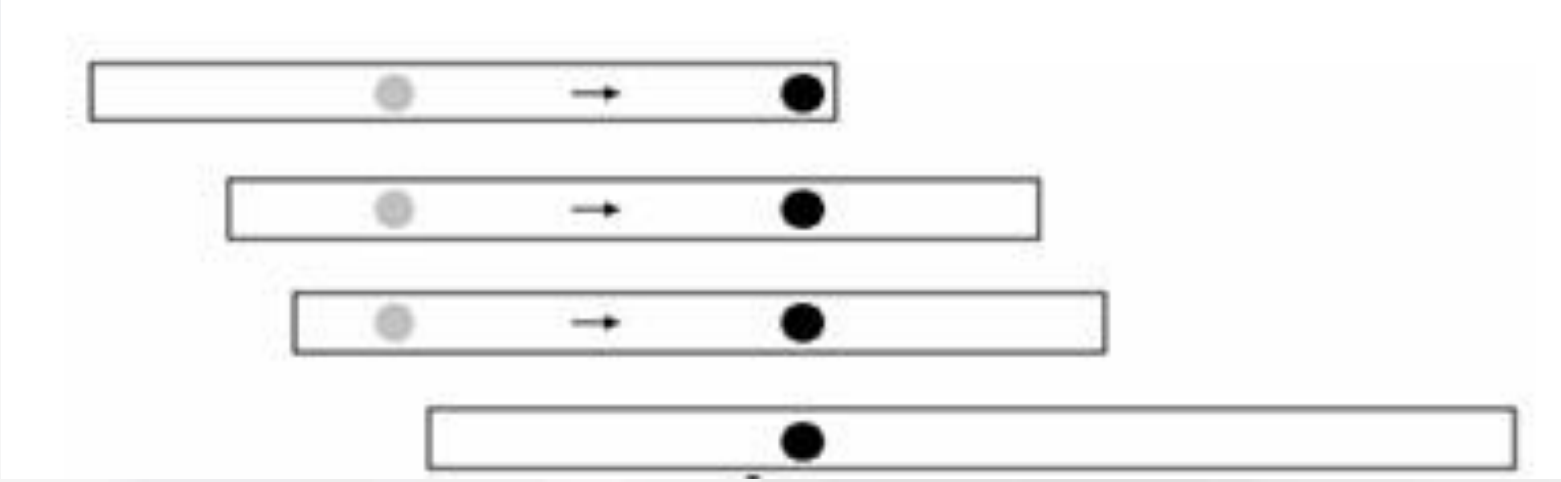
```
2
```

【算法分析】

•算法模型：给n个闭区间[ai,bi]，在数轴上选尽量少的点,使每个区间内至少有一个点。

•算法：首先按b1<=b2<=...<=bn排序。每次标记当前区间的右端点x，并右移当前区间指针，直到当前区间不包含x,再重复上述操作。

•如下图，如果选灰色点，移动到黑色点更优。



【参考程序】

```
#include<iostream>
using namespace std;
int a[10001],b[10001],sum=0,n,m;
void qsort(int x,int y)          //多关键字快排
{
  int i,j,mid1,mid2,t;
  i=x;j=y;mid1=b[(x+y)/2];mid2=a[(x+y)/2];
  while(i<=j)
   { while(b[i]<mid1||((b[i]==mid1)&&(a[i]<mid2)))  ++i;
     while(b[j]>mid1||((b[j]==mid1)&&(a[j]>mid2)))  --j;
     if(i<=j)
      { t=b[j];b[j]=b[i];b[i]=t;
        t=a[j];a[j]=a[i];a[i]=t;
        ++i;  --j;
      }
   }
  if(x<j) qsort(x,j);
  if(i<y) qsort(i,y);
}
```

```
int main()
{
  cin>>n;
  for(int i=1;i<=n;++i)cin>>a[i]>>b[i];
  qsort(1,n);
  for(int i=1,x=-1;i<=n;++i)    //在初始化循环变量的同时，初始化x。
                                //令x=-1可以使第一个区间与其他区间的操作相同。

   {
      if (x>=a[i]) continue;      //如果当前区间包含标记点，就跳过。
      ++sum;   x=b[i];            //更新标记点。
    }
  cout<<sum<<endl;
  return 0;
}
```

【上机练习】

1、删数问题（NOI94）

【问题描述】

键盘输入一个高精度的正整数n(≤240位)，去掉其中任意s个数字后剩下的数字按原左右次序将组成一个新的正整数。编程对给定的n和s，寻找一种方案，使得剩下的数字组成的新数最小。

【输入格式】

n

s

【输出格式】

最后剩下的最小数。

【输入样例】delete.in

175438

4

【输出样例】delete.out

13

2、均分纸牌（NOIP2002）

【问题描述】

有 N 堆纸牌，编号分别为 1，2，…, N。每堆上有若干张，但纸牌总数必为 N 的倍数。可以在任一堆上取若干张纸牌，然后移动。

移牌规则为：在编号为 1 堆上取的纸牌，只能移到编号为 2 的堆上；在编号为 N 的堆上取的纸牌，只能移到编号为 N-1 的堆上；其他堆上取的纸牌，可以移到相邻左边或右边的堆上。

现在要求找出一种移动方法，用最少的移动次数使每堆上纸牌数都一样多。

例如 N=4，4 堆纸牌数分别为：　①　9　②　8　③　17　④　6

移动3次可达到目的：

从 ③ 取4张牌放到④（9 8 13 10）->从③取3张牌放到 ②（9 11 10 10）->从②取1张牌放到①（10 10 10 10）。

【输入格式】

N（N 堆纸牌，1 <= N <= 100）

A1 A2 … An （N 堆纸牌，每堆纸牌初始数，l<= Ai <=10000）

【输出格式】

所有堆均达到相等时的最少移动次数。

【样例输入】(Playcard.in)

```
4
9 8 17 6
```

【样例输出】(Playcard.out)

```
3
```

3、拦截导弹问题（NOIP1999）

【问题描述】

某国为了防御敌国的导弹袭击，开发出一种导弹拦截系统，但是这种拦截系统有一个缺陷：虽然它的第一发炮弹能够到达任意的高度，但是以后每一发炮弹都不能高于前一发的高度。某天，雷达捕捉到敌国的导弹来袭，由于该系统还在试用阶段。所以一套系统有可能不能拦截所有的导弹。

输入导弹依次飞来的高度（雷达给出的高度不大于30000的正整数）。计算要拦截所有导弹最小需要配备多少套这种导弹拦截系统。

【输入格式】

n颗依次飞来的高度（1≤n≤1000）.

【输出格式】

要拦截所有导弹最小配备的系统数k。

【输入样例】missile.in

389 207 155 300 299 170 158 65

【输出样例】missile.out

2

4、排队接水(water.pas)

【问题描述】

有n个人在一个水龙头前排队接水，假如每个人接水的时间为Ti，请编程找出这n个人排队的一种顺序，使得n个人的平均等待时间最小。

【输入格式】

输入文件共两行，第一行为n；第二行分别表示第1个人到第n个人每人的接水时间T1，T2，…，Tn，每个数据之间有1个空格。

【输出格式】

输出文件有两行，第一行为一种排队顺序，即1到n的一种排列；第二行为这种排列方案下的平均等待时间(输出结果精确到小数点后两位)。

【输入输出样例】

water.in

```
10
3 2 7 8 1 4 9 6 10 5
56 12 1 99 1000 234 33 55 99 812
```

water.out

```
532.00
```

5、最大整数（Noip1998连接多位数）

【问题描述】

设有n个正整数（n≤20），将它们联接成一排，组成一个最大的多位整数。

例如：n=3时，3个整数13，312，343联接成的最大整数为：34331213

又如：n=4时，4个整数7，13，4，246联接成的最大整数为：7424613

【输入格式】

n

n个数

【输出格式】

联接成的多位数

【输入样例】maxnum.in

3

13 312 343

【输出样例】maxnum.out

34331213

6、纪念品分组(NOIP2007)

【题目描述】

元旦快到了，校学生会让乐乐负责新年晚会的纪念品发放工作。为使得参加晚会的同学所获得的纪念品价值相对均衡，他要把购来的纪念品根据价格进行分组，但每组最多只能包括两件纪念品，并且每组纪念品的价格之和不能超过一个给定的整数。为了保证在尽量短的时间内发完所有纪念品，乐乐希望分组的数目最少。

你的任务是写一个程序，找出所有分组方案中分组数最少的一种，输出最少的分组数目。

【输入格式】

输入文件group.in包含*n*+2行：

第1行包括一个整数*w*，为每组纪念品价格之和的上限。

第2行为一个整数*n*，表示购来的纪念品的总件数。

第3~*n*+2行每行包含一个正整数*pi* (5 <= *pi* <= *w*)，表示所对应纪念品的价格。

【输出格式】

输出文件group.out仅一行，包含一个整数，即最少的分组数目。

【输入输出样例】

group.in
100 9 90 20 20 30 50 60 70 80 90

group.out
6

【限制】

50%的数据满足：1 <= *n* <= 15

100%的数据满足：1 <= *n* <= 30000, 80 <= *w* <= 200

7、合并果子(Noip2004)

【问题描述】在一个果园里，多多已经将所有的果子打了下来，而且按果子的不同种类分成了不同的堆。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。可以看出，所有的果子经过n-1次合并之后，就只剩下一堆了。多多在合并果子时总共消耗的体力等于每次合并所耗体力之和。

因为还要花大力气把这些果子搬回家，所以多多在合并果子时要尽可能地节省体力。假定每个果子重量都为1，并且已知果子的种类数和每种果子的数目，你的任务是设计出合并的次序方案，使多多耗费的体力最少，并输出这个最小的体力耗费值。

例如有3种果子，数目依次为1，2，9。可以先将 1、2堆合并，新堆数目为3，耗费体力为3。接着，将新堆与原先的第三堆合并，又得到新的堆，数目为12，耗费体力为 12。所以多多总共耗费体力=3+12=15。可以证明15为最小的体力耗费值。

【输入文件】

输入文件**fruit.in**包括两行，第一行是一个整数n（$1 \leq n \leq 10000$），表示果子的种类数。第二行包含n个整数，用空格分隔，第i个整数ai（$1 \leq ai \leq 20000$）是第i种果子的数目。

【输出文件】

输出文件**fruit.out**包括一行，这一行只包含一个整数，也就是最小的体力耗费值。输入数据保证这个值小于231。

【样例输入】
3
1 2 9
【样例输出】
15
【数据规模】
对于30%的数据，保证有n <= 1000；
对于50%的数据，保证有n <= 5000；
对于全部的数据，保证有n <= 10000。

8、美元汇率（DOLLARS.PAS）

【问题描述】

在以后的若干天里戴维将学习美元与德国马克的汇率。编写程序帮助戴维何时应买或卖马克或美元，使他从100美元开始，最后能获得最高可能的价值。

【输入格式】

输入文件的第一行是一个自然数N，1≤N≤100，表示戴维学习汇率的天数。

接下来的N行中每行是一个自然数A，1≤A≤1000。第i+1行的A表示预先知道的第i+1天的平均汇率，在这一天中，戴维既能用100美元买A马克也能用A马克购买100美元。

【输出格式】

输出文件的第一行也是唯一的一行应输出要求的钱数(单位为美元，保留两位小数)。

注意：考虑到实数算术运算中进位的误差，结果在正确结果0.05美元范围内的被认为是正确的，戴维必须在最后一天结束之前将他的钱都换成美元。

【输入样例】DOLLARS.IN

```
5
400
300
500
300
250
```

【输出样例】DOLLARS.OUT

```
266.66
```

样例解释 (无需输出)

Day 1 ... changing 100.0000 美元= 400.0000 马克
Day 2 ... changing 400.0000 马克= 133.3333 美元
Day 3 ... changing 133.3333 美元= 666.6666 马克
Day 5 ... changing 666.6666 马克= 266.6666 美元

9、零件分组（stick.pas）

【问题描述】

某工厂生产一批棍状零件，每个零件都有一定的长度（Li）和重量（Wi）。现在为了加工需要，要将它们分成若干组，使每一组的零件都能排成一个长度和重量都不下降（若i<j，则Li<=Lj，Wi<=Wj）的序列。请问至少要分成几组?

【输入格式】

第一行为一个整数N（N<=1000），表示零件的个数。第二行有N对正整数，每对正整数表示这些零件的长度和重量，长度和重量均不超过10000。

【输出格式】

仅一行，即最少分成的组数。

【输入样例】STICK.IN

```
5
8 4 3 8 2 3 9 7 3 5
```

【输出样例】STICK.OUT

```
2
```