二分与三分算法

信智学院　陈琰宏

教学内容

01 整数二分

02 小数二分

03 三分

04 案例讲解

一、二分

二分法在一个单调有序的集合或函数中查找一个解，每次分为左右两部分，判断解在哪个部分中并调整上下界，直到找到目标元素，每次二分后都将舍弃一半的查找空间，因此效率很高。

若求解的问题的定义域为整数域，对于长度为N的求解区间，算法需要logn次来确定出分界点。

对于定义域在实数域上的问题，可以类似于上面的方法，判断R-L的精度是否达到要求，即R-L>=eps。

如果我们指定二分的次数t，那么对于初始的求解区间长度L，算法结束后的r-l值会为L/2^t。

二分算法的复杂度O(二分次数*单次判定复杂度)。

二分的经典写法（整数）

```
int check(int a[],int x){

    int l=1,r=n;
    while(l<=r){
        int mid=(l+r)>>1;
        if(a[mid]<=x)
            l=mid+1,a
        else//>= 右侧
            r=mid-1;
    }
    return ans;
}
```

```
while(le<=ri){
    int mid=(le+ri)/2;
    if(check(mid)){
        ans=mid;
        le=mid+1;
    }
    else
        ri=mid-1;
}
```

二分的经典写法（小数）

```
1  int Erfen(double l,double r){
2      double mid;
3      while(fabs(l-r)>dlt){
4          mid=(l+r)/2.0
5          if(check(mid))
6              r=mid;
7          else
8              l=mid;
9      }
10     return l;
11 }
```

二、二分法常见模型

1.二分答案

最小值最大（或是最大值最小）问题，这类双最值问题常常使用二分法求解，也就是确定答案后，配合贪心、DP等其它算法来检验这个答案是否合法，将最优化问题转变为判定性问题。例如：将长度为n的序列ai分成最多m个连续段，求所有分法中每段和的最大值最小是多少。

2.二分查找

具有单调性的布尔表达式求解分界点，比如在有序数列中求数字 的排名。

三、典型例题

1 [3630] 愤怒的牛

农夫 John 建造了拥有N (2 <= N <= 100,000)个牛舍的小屋，这些牛舍排在一条直线上。这些牛舍依次编号为x1,...,xN (0 <= xi <= 1,000,000,000). 但是，John的C (2 <= C <= N)头牛们并不喜欢这种布局，而且几头牛放在一个牛舍里，它们会相互攻击。为了不让牛互相伤害，John决定自己给牛分配舍，使任意两头牛之间的最小距离尽可能的大。求最大的最小距离是什么。

输入第一行：空格分隔的两个整数N和C，第2-n+1行：i+1行指出了xi的位置输出一个整数，最大的最小

样例输入

```
5 3
1 2 8 4 9
```

样例输出

```
3
```

题解：

（1）暴力法。从大到小枚举最小距离的值dis，然后检查，如果发现有一次不行，那么上次枚举的就是最大值。如何检查呢？

用贪心法：

1. 对牛舍的位置x进行排序
2. 把第一头牛放入x0的牛舍
3. 如果第i头牛放入了xj的话，第i+1头牛就要放入满足xj+d≤xk的最小的xk中

第一头牛放在x0，第二头牛放在xj≥x0+dis的点xi,第三头牛放在xk≥xj+dis的点xk。如果在当前最小距离下，不能放c条牛，那么这个dis就不可取。复杂度O(nc)。

2）二分。分析从大到小检查dis的过程，发现可以用二分的方法找这个dis。这个dis符合二分法：它有上下边界、它是单调递增的。复杂度O(nlogn)。

```cpp
#include<bits/stdc++.h>
using namespace std;
int n,c,x[100005];//牛棚数量，牛数量，牛棚坐标

bool check(int dis){      //当牛之间距离最小为dis时，检查牛棚够不够
    int cnt=1, place=0;  //第1头牛，放在第1个牛棚
    for (int i = 1; i < n; ++i)      //检查后面每个牛棚
        if (x[i] - x[place] >= dis){ //如果距离dis的位置有牛棚
            cnt++;       //又放了一头牛
            place = i;  //更新上一头牛的位置
        }
    if (cnt >= c) return true;    //牛棚够
    else          return false;  //牛棚不够
}

```

```
int main(){
    scanf("%d%d",&n, &c);
    for(int i=0;i<n;i++)    scanf("%d",&x[i]);
    sort(x,x+n);                    //对牛棚的坐标排序
    int left=0, right=x[n-1]-x[0];  //R=1000000也行，因为是Log(n)的，很快
                                    //优化：把二分上限设置为1e9/c
    int ans = 0;
    while(left <= right){
        int mid = left + (right - left)/2;      //二分
        if(check(mid)){             //当牛之间距离最小为mid时，牛棚够不够?
            ans = mid;              //牛棚够，先记录mid
            left = mid + 1;         //扩大距离
          }
        else
            right = mid-1;          //牛棚不够，缩小距离
    }

    cout << ans;                    //打印答案
    return 0;
}
```

[3635] Best Cow Fences 最佳牛围栏

农夫约翰的农场由 N （1 <= N <= 100,000）块田地组成，每块地里都有一定数量的牛，其数量不会少于 1 头，也不会超过 2000 头。约翰希望用围栏将一部分连续的田地围起来，并使得围起来的区域内每块地包含的牛的数量的平均值达到最大。

围起区域内至少需要包含 F 块地，在给定条件下，计算围起区域内每块地包含的牛的数量的平均值可能的最大值是多少。

输入第一行输入整数 N 和F，数据间用空格隔开。 N （1 <= N <= 100,000）接下来 N 行，每行输入一个整数，第 i+1 行输入的整数代表第 i 片区域内包含的牛的数目。输出一个整数，表示平均值的最大值乘以 1000 再 向下取整 之后得到的结果。 1≤N≤100000 1≤F≤N

分析

该题可以抽象为给定一个长度为n的正整数序列A，求一个平均数最大的，长度不小于L的子段。

3 5 1 0 9 4 5 12 3 6 5 12 9

枚举每一个L [F,n]的长度，确定最大平均值。时间复杂度是 O(n^2)。因此我们要寻找更高效的处理方式使时间复杂度降低。

考虑L的范围是[F,n]，并且是一个单调区间，可以考虑对L进行二分枚举。

无效的二分，因为要对每个L求平均值，该平均值不是单调的!

分析

题目求“最大的平均值”，考虑对平均值做二分，因为平均值的范围在[1,2000]。

如何判断该平均值符合题意，即 check函数:

① 如果把数列中每个数都减去二分的值，问题就转化为判定“是否存在一个长度不小于L的子段，子段和非负”。即求一个子段，它的和最大，子段的长度不小于L。

② 问题转化为求最大连续字段和问题。

[2366] 求最大和

给定一个长度为n的整数序列，求一个和最大的，
长度不小于L的连续子序列和。
输入n个整数序列
输出最大和。

8 3
1 -2 5 2 1 3 -6 20

分析

求一个子序列，它的和最大，子序列的长度不小于L.
子段和可以转化为前缀和相减的形式，即设sumi表示A1~Ai的和，则有：

$$\max_{i-j\geq L}\{A_{j+1}+A_{j+2}+..+A_i\}=$$

$$\max_{L\leq i\leq n}\{sum_i-\min_{0\leq j\leq i-L}\{sum_j\}\}$$

仔细观察上面的式子可以发现，随着i的增长，j的取值范围0~i-L每次只会增大1。换言之，每次只会有一个新的取值进入min{sum}的候选集合，所以我们没有必要每次循环枚举j，只需要用一个变量记录当前最小值，每次与新的取值sumf-L取min 就可以了。

10 3
1 -2 5 2 -7 3 -6 20 -4 5

i	0	1	2	3	4	5	6	7	8	9	10
a[i]		1	-2	5	2	-7	3	-6	20	-4	5
sum[i]	0	1	-1	4	6	-1	2	-4	16	12	17
minV	在离当前sum[i]前找到距离>=L的最小值 sum[j]										
minV				0	0	-1	-1	-1	-1	-1	-4
ans				4	6	6	6	6	17	17	21

```
int ans = -1e9, min_val = 1e9;
for(int i=L;i<=n;i++) {
    min_val = min(min_val, sum[i - L]);
    ans = max(ans, sum[i] - min_val);
}
```

```
 3  const int N = 1e6 + 10;
 4  int a[N], b[N], sum[N];
 5  int n, L;
 6
 7  int main() {
 8      cin >> n >> L;
 9      for(int i=1;i<=n;i++) {
10          cin >> a[i];
11          sum[i]=sum[i-1]+a[i];
12      }
13      int ans = -1e9, min_val = 1e9;
14      for(int i=L;i<=n;i++) {
15          min_val = min(min_val, sum[i - L]);
16          ans = max(ans, sum[i] - min_val);
17      }
18      cout << ans;
19      return 0;
20  }
```

[3635] 最佳牛围栏 代码

```
20  int main() {
21      scanf("%d %d", &n, &m);
22      double l = 0, r = 0;
23      for (int i = 1; i <= n; i++) {
24          scanf("%d", &cows[i]);
25          r = max(r, (double)cows[i]);
26      }
27
28      while(r - l >= 1e-5) {
29          double mid = (l + r) / 2;
30          if(check(mid)) l = mid;
31          else r = mid;
32      } printf("%d\n", (int)(r * 1000));
33      return 0;
34  }
```

```
 4  const int N = 100005;
 5  int cows[N]; double sum[N];
 6  int n, m;
 7  bool check(double avg) {
 8      for (int i = 1; i <= n; i++) {
 9          //计算前缀和
10          sum[i] = sum[i - 1] + cows[i] - avg;
11      }
12      double minv = 0; //设置最小值
13      for (int i = 0, j = m; j <= n; j++, i++) {
14          minv = min(minv, sum[i]); //找最优极小值
15          if(sum[j] - minv >= 0) return true; //进行判断
16      } return false;
17      //如果所有的都不满足，那么这个平均数就一定不满足
18  }
```

[2792] 一元三次方程求解

有形如：$ax^3+bx^2+cx+d=0$这样的一个一元三次方程。给出该方程中各项的系数（a,b,c,d均为实数），并约定该方程存在三个不同实根（根的范围在-100至100之间），且根与根之差的绝对值≥1。

要求由小到大依次在同一行输出这三个实根（根与根之间留有空格），并精确到小数点后2位。

提示：记方程f（x）=0，若存在2个数x1和x2，且x1<x2，f（x1）*f（x2）＜0，则在（x1，x2）之间一定有一个根。

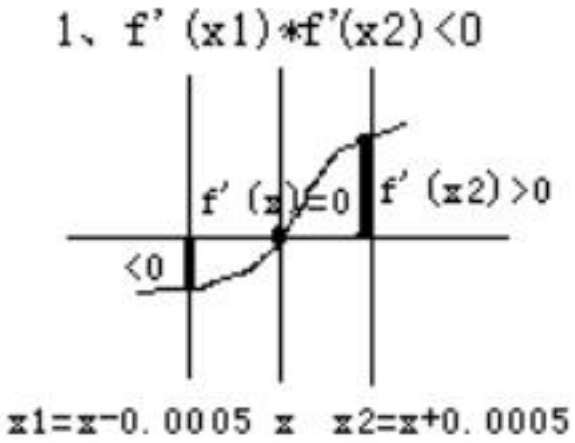
输入：a，b，c，d

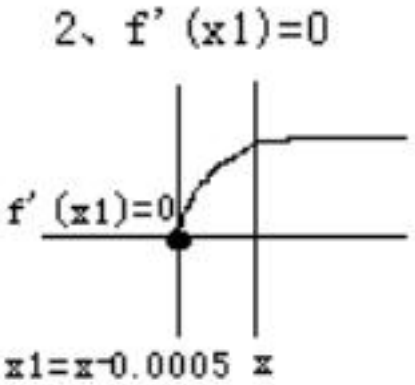
输出：三个实根（根与根之间留有空格）

【输入输出样例】

输入：1 -5 -4 20

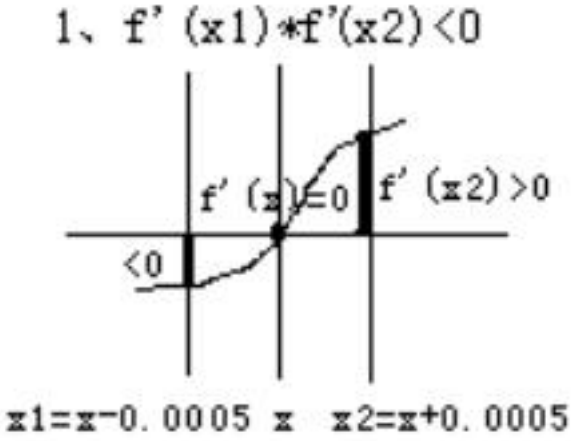
输出：-2.00 2.00 5.00

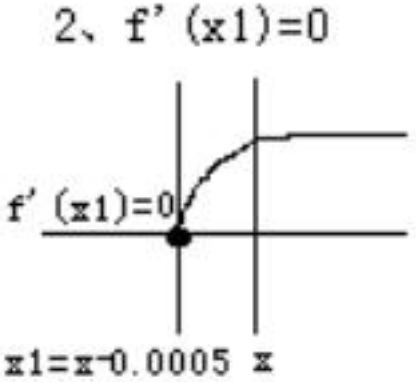




算法分析

这是一道有趣的解方程题。为了便于求解，设方程f(x)=ax3+bx2+cx+d＝0，设根的值域（-100至100之间）中有x， 其左右两边相距0.0005的地方有x1和x2两个数，即x1=x-0.0005，x2=x+0.0005。x1和x2间的距离（0.001）满足精度要求（精确到小数点后2位）。若出现如图1所示的两种情况之一，则确定x为f(x)=0的根。





有两种方法计算f(x)=0的根x:

1．枚举法

根据根的值域和根与根之间的间距要求(≥1)，我们不妨将根的值域扩大100倍(-10000≤x≤10000)，依次枚举该区间的每一个整数值x，并在题目要求的精度内设定区间：x1=　,x2=　。若区间端点的函数值f(x1)和f(x2)异号或者在区间端点x1的函数值f(x1)=0，则确定为f(x)=0的一个根。由此得出算法：

```
输入方程中各项的系数a，b，c，d ；
for (x=-10000;x<=10000;x++)
//枚举当前根*100的可能范围
  {
        x1=(x-0.05)/100;
        x2=(x+0.05)/100;
//在题目要求的精度内设定区间
if (f(x1)*f(x2)<0||f(x1)==0)
        printf(“%.2f”,x/100);
//若在区间两端的函数值异号
或在x1处的函数值为0，
则确定x/100为根

}
```

2. 分治法

枚举根的值域中的每一个整数x(-100≤x≤100)。由于根与根之差的绝对值≥1，因此设定搜索区间[x1，x2]，其中x1=x，x2=x+1。若

(1)f(x1)=0，则确定x1为f(x)的根；

(2)f(x1)*f(x2)>0，则确定根x不在区间[x1，x2]内，设定[x2，x2+1]为下一个搜索区间

(3)f(x1)*f(x2)<0，则确定根x在区间[x1，x2]内。

如果确定根x在区间[x1，x2]内的话（f(x1)*f(x2)<0），如何在该区间找到根的确切位置。采用二分法，将区间[x1，x2]分成左右两个子区间：左子区间[x1，x]和右子区间[x，x2]（其中x=(x1+x2)/2）：

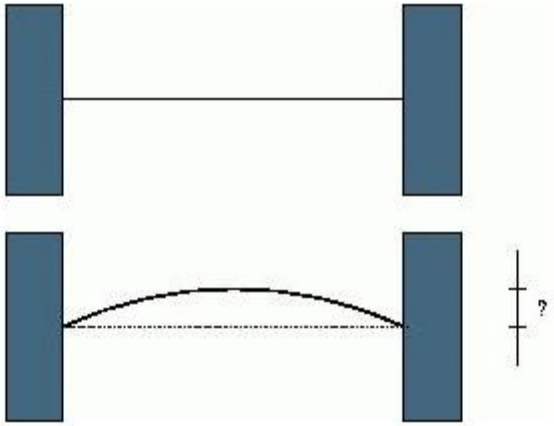
如果f(x1)*f(x)≤0，则确定根在左区间[x1，x]内，将x设为该区间的右指针（x2=x），继续对左区间进行对分；如果f(x1)*f(x)>0，则确定根在右区间[x，x2]内，将x设为该区间的左指针（x1=x），继续对右区间进行对分；

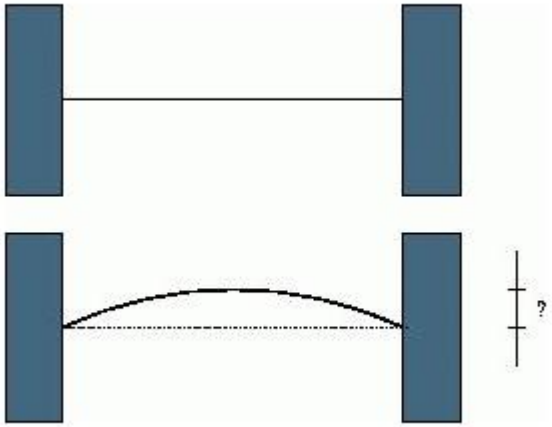
上述对分过程一直进行到区间的间距满足精度要求为止（x2-x1<0.001）。此时确定x1为f(x)的根。

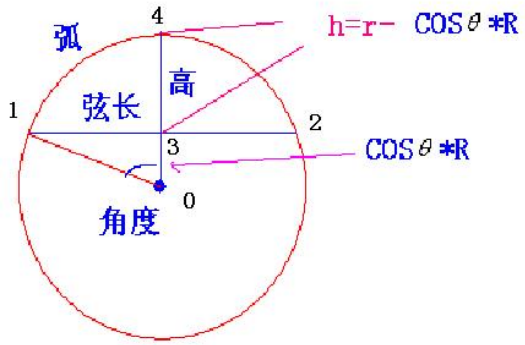
```cpp
#include<bits/stdc++.h>
using namespace std;
double a,b,c,d;
double f(double x){
    return a*x*x*x+b*x*x+c*x+d;
}
int main(){
    while(cin>>a>>b>>c>>d){
        for(double i=-100;i<=100;i++){////枚举每一个可能的根
            double x=i,y=i+1;////确定根的可能区间
            if(f(x)==0) //若x1为根，则输出
                printf("%.2lf ",x);
            else if(f(x)*f(y)<0){ //若根在区间[x，y]中
                while(y-x>=0.001){
                //若区间[x1，x2]不满足精度要求，则循环
                    double mid=(x+y)/2; //计算区间[x1，x2]的中间位置
                    if(f(x)*f(mid)<=0)
                        y=mid;//若根在左区间，则调整右指针
                    else x=mid;//若根在右区间，则调整左指针
                }
                printf("%.2lf ",x);//区间[x1，x2]满足精度要求，确定x为根
            }
        }
        printf("\n");
    }
    return 0;
}
```

[3226] 热膨胀

棒一根，加热膨胀成圆弧，求膨胀后的高度。
已知：棒的长度L，膨胀系数C，加热温度n.
L'=(1+n*C)*L







分析：

h=r-r*cos θ

*关键求 $*\theta$

ll/2=r* θ (1式)

l/2=r*sin θ (2式)

1式/2式，得到 θ 的方程

ll/l = θ / sin θ

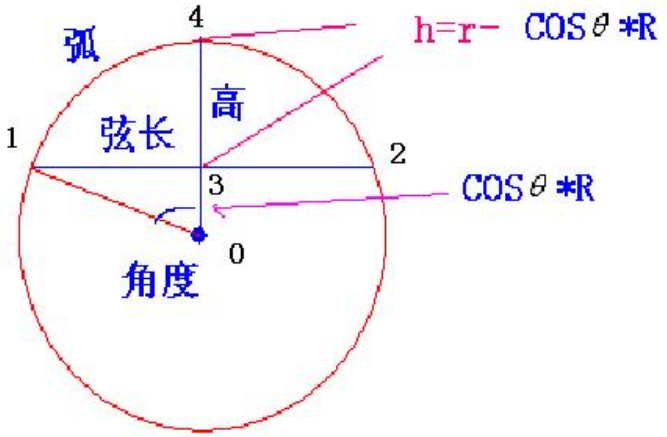
能解吗？怎么解？

解方程：插值法 二分法

$$ll/l = \theta / \sin\theta = t$$

$$\sin\theta = \theta * t$$

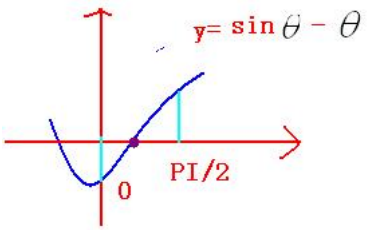
$$y = \sin\theta - t * \theta$$



对1/4圆弧来说，确定 θ的最大值、最小值

最大值：PI/2

最小值：0



```c
#include <stdio.h>
#include <math.h>
int main()
{
 double l,c,n,ll,t,min,max,mid,y,h;
 while(scanf("%lf %lf %lf",&l,&n,&c)!=-1)
 {//l是弦长 ll是弧长 min,max,mid表示角度 h是高度
  if(l==-1&&n==-1&&c==-1)
   break;
  if(n*c==0)
   printf("%.3lf\n",0.0);
  else
  {
 }
 return 0;
}
```

```
12      else
13      {
14          ll=(1+c*n)*l;
15          t=l/ll;
16          min=0;
17          max=acos(-1)/2;//当是半圆时PI*r=ll; l=2r;l和ll的最大比值为PI/2
18
19          for(int i=0;i<100;i++)
20          {
21              mid=(min+max)/2;
22              y=sin(mid)-t*mid;//求角度，当角度很小的时候
23              if(y>0)
24               min=mid;
25              else
26               max=mid;
27          }
28          h=(1-cos(mid))*ll/2/mid;
29          /*   半径r; 圆弧长ll;   ll=2*角度mid*r;
30          h=r-rcos角度=(1-cos角度)× r;   r=ll/2/mid   */
31          printf("%.3lf\n",h);
32      }
33  }
```

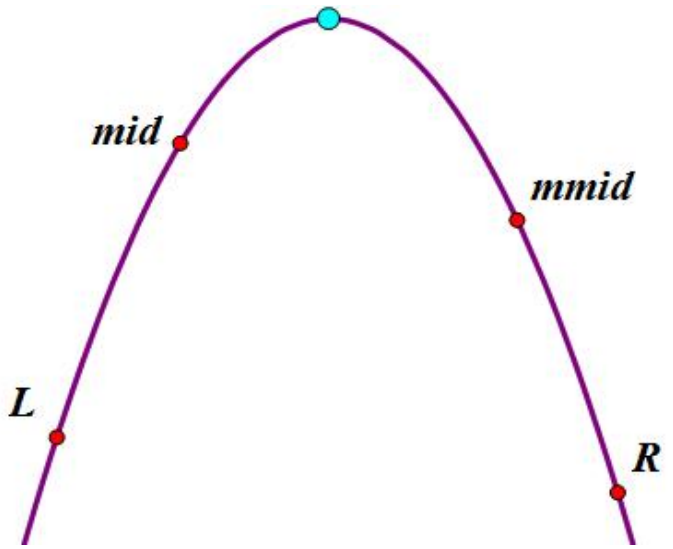
四、三分

三分法适用于求解凸性函数的极值问题，二次函数就是一个典型的单峰函数。

三分法与二分法一样，它会不断缩小答案所在的求解区间。二分法缩短区间利用的原理是函数的单调性，而三分法利用的则是函数的单峰性。

已知：左右端点L、R，要求找到上图中空心点的位置

思路：通过不断缩小［L,R］的范围，无限逼近空心点

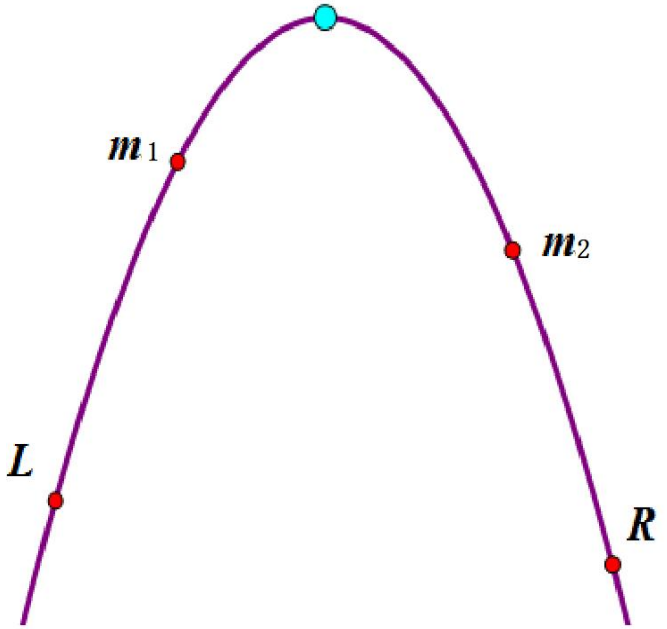


四、三分

设当前求解的区间为[l,r]，令

m1=l+(r-l)/3，m2=r-(r-l)/3，

接着我们计算这两个点的函数值f(m1)，f(m2)，之后我们将两点中函数值更优的那个点称为好点（若计算最大值，则更大的那个点就为好点，计算最小值同理），而函数值较差的那个点称为坏点。



我们可以证明，最优点与好点会在坏点同侧。例如，f(m1)>f(m2)，则是m1好点，而m2是坏点，因此最后的 最优点会与m1一起在 m2的左侧，即我们的求解区间由变为了[1,m2]。因此根据这个结论我们可以不停缩短求解区间，直至可以得出近似解。

与二分一样，我们可以指定三分的次数，或是根据r-l的值来终止算法。

```
double l=0,r=1e9;
while(r-l>=1e-3){
    double m1=l+(r-l)/3;
    double m2=r-(l-r)/3;
    if(f(m1)<f(m2))
        l=m1;
    else
        r=m2;
}
```

注意，我们在介绍单峰函数时特别强调了“严格”单调性。

若在三分过程中遇到 f(m1) = f(m2)，当函数严格单调时，令 l = m1或 r = m2均可。如果函数不严格单调，即在函数中存在一段值相等的部分，那么我们无法判断定义域的左右边界如何缩小，三分法就不再适用。

[3638] Curves

明明做作业的时候遇到了n个二次函数

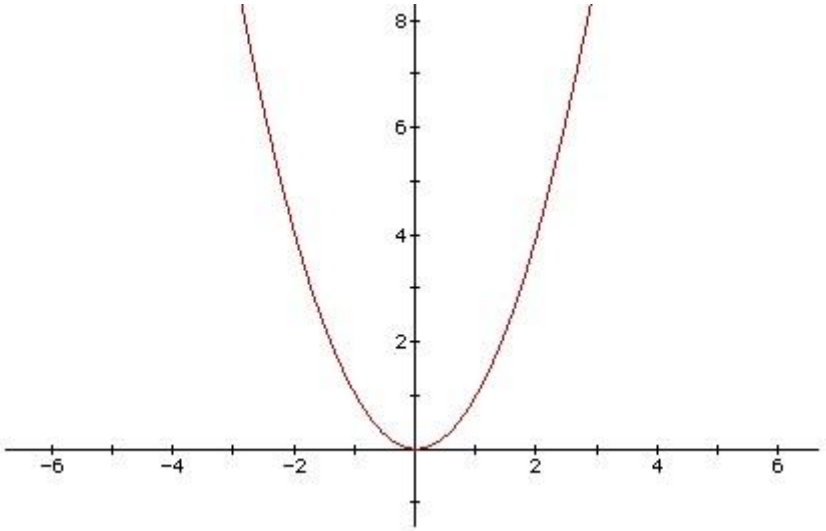
$S_i(x) = ax^2 + bx + c$,

他突发奇想设计了一个新的函数

$F(x) = \max(S_i(x))$,

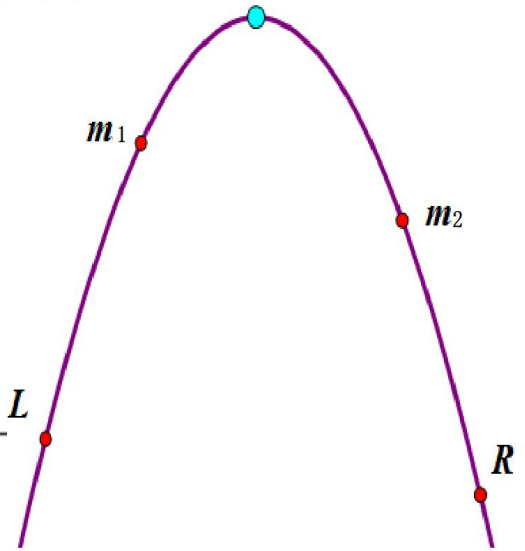
$i = 1...n$.

明明现在想求这个函数在[0,1000]的最小值，要求精确到小数点后四位四舍五入。



[3638] Curves

```
#include<bits/stdc++.h>
using namespace std;
int n;
double a[210000],b[210000],c[210000];
double check(double x)
{
    double maxx=a[1]*x*x+b[1]*x+c[1];
    for(int i=2;i<=n;i++)
        maxx=max(maxx,a[i]*x*x+b[i]*x+c[i]);
    return maxx;
}
int main()
{
    int t;
    cin>>t;
    while(t--)
    {
    return 0;
}
```



[3638] Curves

```
while(t--)
{
    cin>>n;
    for(int i=1;i<=n;i++){
        cin>>a[i]>>b[i]>>c[i];
    }
    double l=0,r=1000;//总的区间就是0-1000
    while(r-l>=1e-9)
    {
        double mid1=l+(r-l)/3.0,mid2=r-(r-l)/3.0;
        if(check(mid1)>check(mid2))
            l=mid1;//如果左边比右边大，说明小的值在右边，去右边找
        else
            r=mid2;//否则去左边
        //最后还是要更新mid1和mid2的值
    }
    printf("%.4lf\n",check(l));//最后判断这个l的值
}
```

[3633] 数列分段II Section II

对于给定的一个长度为N的正整数数列 A，现要将其分成M(M≤N)段，并要求每段连续，且每段和的最大值最小。

关于最大值最小：

例如一数列 4 2 4 5 1要分成 3 段

将其如下分段：[4 2][4 5][1]

第一段和为 6 ，第 2 段和为 9 ，第 3 段和为 1 ，和最大值为 9 。

将其如下分段：[4][2 4][5 1]

第一段和为 4 ，第 2 段和为 6 ，第 3 段和为 6 ，和最大值为 6 。并且无论如何分段，最大值不会小于 6 。

所以可以得到要将数列 4 2 4 5 1要分成 3段，每段和的最大值最小为 6 。

[3633] 数列分段II Section II

输入第 1 行包含两个正整数N，M。第 2 行包含 N 个空格隔开的非负整数 A_i，含义如题目所述。

输出一个正整数，即每段和**最大值最小**为多少。

输入样例

5 3

4 2 4 5 1

输出样例

6

对于 100% 的数据，有 N≤100000,M≤N,Ai 之和不超过 10^9。

分析

当最大值最小，最小值最大，看到这类字眼就要想到二分答案。

二分答案，顾名思义，答案是什么就二分什么。答案要求的是每段和的最大值最小是什么，那就二分每段和的最大值。

那么check(x)，就相当于check在每一段的和都不超过x的情况下，能否把数组切成最多m段，如果能切成≤m段又满足条件，那这个解是可行解，就继续二分看有没有更优解。

分析：如何check

如果不让计算机来做这个问题，让你来切这个数组，你会怎么切？

这里的贪心思想比较简单，因为是切成的段要求是连续的，因此切数组的最优步骤：

1. 按顺序考虑每一个数，维护段数，维护当前在考虑的这个段的数的和

2. 如果当前数加入当前段不超过限制x，把这个数加入这个段，继续考虑下一个数

3. 否则，说明需要新开一段，把当前数加入新段，段数加一

4. 最后，看看我们切了几刀，如果切成的段数不超过m，那么return true

分析

```
3   const int N = 1e5 + 10;
4   int a[N];
5   int n, m;
6   bool check(int x)
7   // 在每一段的和都不超过x的情况下，能否把数组切成最多m段
8   {
9       int seg = 0; // seg即segment,维护段数
10      int sum = 0; // sum维护当前在考虑的段，其中的数的和
11      for (int i = 1; i <= n; i++)
12      {//贪心在这里：不超过当x，就一直扩充这一段
13          if (sum + a[i] <= x)
14              sum += a[i];
15          else //当前这个数不能加进上一段了，必须新开一段
16          {
17              sum = a[i]; //这个数加入新的段
18              seg++;      //段数加一
19          }
20      }
21      return seg < m;//符合要求1 否则0
22  }
```

分析

```
23  int main()
24  {
25      cin >> n >> m;
26      int l = 0, r = 0, ans = 0;
27      for (int i = 1; i <= n; i++)
28      {
29          scanf("%d", &a[i]);
30          l = max(l, a[i]); //答案的下界是a数组中的最大值
31          r += a[i];//答案的上界是a数组中所有数的和
32              //(相当于分成了一整段)
33      }
34      while (l <= r) //二分模板
35      {
36          int mid = ( l + r ) >> 1;
37          if (check(mid))
38              ans=mid,r = mid-1;
39          else
40              l = mid + 1;
41      }
42      cout << ans << endl;
43      return 0;
44  }
```

[3636] 灯泡

相比 wildleopard 的家，他的弟弟 mildleopard 比较穷。他的房子是狭窄的而且在他的房间里面仅有一个灯泡。每天晚上，他徘徊在自己狭小的房子里，思考如何赚更多的钱。

有一天，他发现他的影子的长度随着他在灯泡和墙壁之间走到时发生着变化。 一个突然的想法出现在脑海里，他想知道他的影子的最大长度。

[3636] 灯泡

输入第一行包含一个整数 T，表示测试数据的组数。对于每组测试数据，仅一行，包含三个实数 H，h 和 D，H 表示灯泡的高度，h 表示 mildleopard 的身高，D 表示灯泡和墙的水平距离。

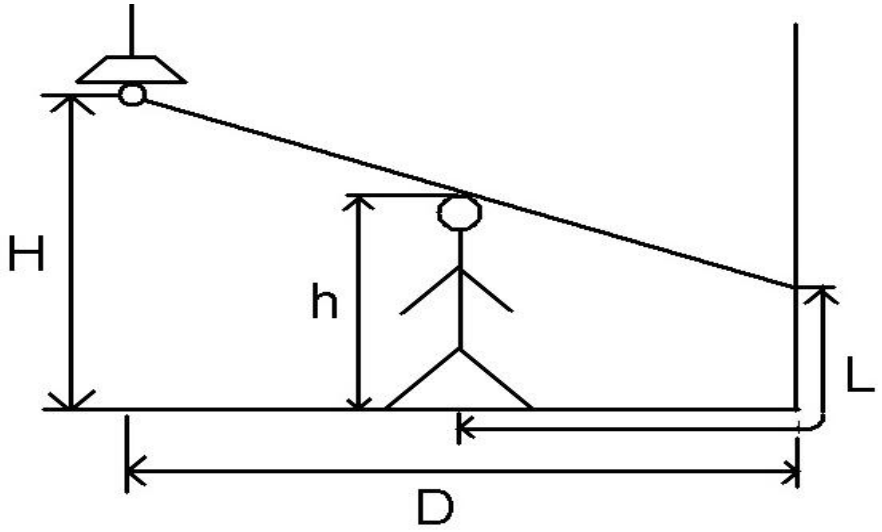
输出共 T 行，每组数据占一行表示影子的最大长度，保留三位小数。

输入样例

```
3
2 1 0.5
2 0.5 3
4 3 4
```

输出样例

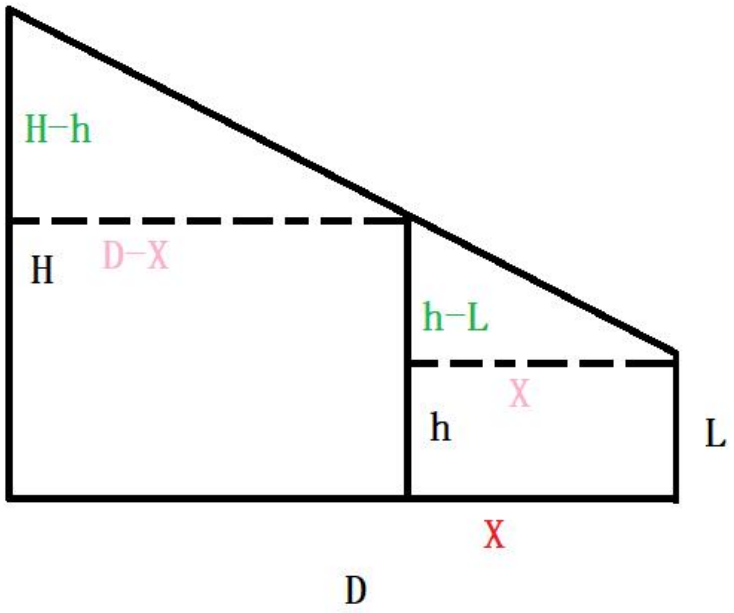
```
1.000
0.750
4.000
```



分析

不难得出，设人和墙的距离为x，影长可以化为

$$x - D + \frac{D * (H - h)}{x - D} + H + D$$

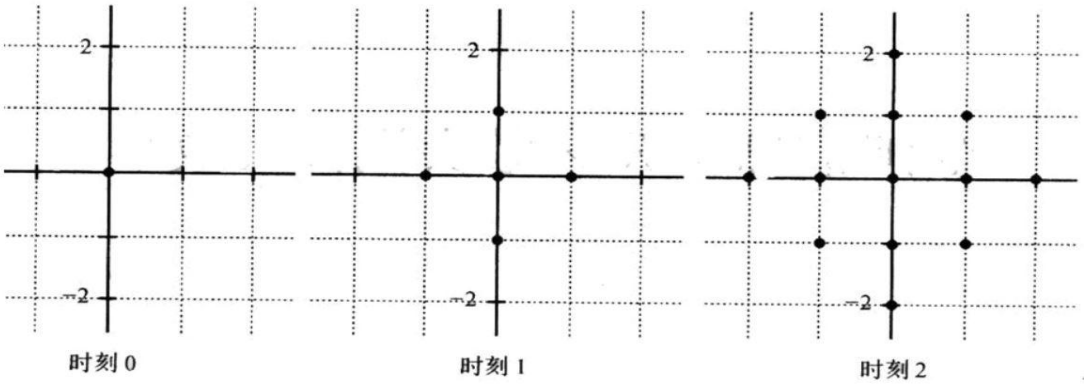


分析

[3632] 扩散

一个点每过一个单位时间就会向四个方向扩散一个距离。两个点a、b连通，记作e(a,b),当且仅当a、b的扩散区域有公共部分。连通块的定义是块内的任意两个点u、v都必定存在路径e(u,a0),e(a0,a1),…,e(ak,v)。给定平面上的n给点，问最早什么时刻它们形成一个连通块。 1≤X[i],Y[i]≤10^9

输入第一行一个数n，以下n行，每行一个点坐标。输出一个数，表示最早的时刻所有点形成连通块。1≤N≤50;



分析

根据题意，每个点每过一个单位时间就会向四个方向扩散一个距离。我们使用输入样例来分析输出结果。开始我们有两个点 （0, 0）和（5, 5）。时间和扩散点坐标如下表。

时间	A 点				B 点			
t=0	(0, 0)				(5, 5)			
	上	左	下	右	上	左	下	右
t=1	(0, 1)	(1, 0)	(0, -1)	(-1, 0)	(5, 6)	(6, 5)	(5, 4)	(4, 5)
t=2	(0, 2)	(2, 0)	(0, -2)	(-2, 0)	(5, 7)	(7, 5)	(5, 3)	(3, 5)
t=3	(0, 3)	(3, 0)	(0, -3)	(-3, 0)	(5, 8)	(8, 5)	(5, 2)	(2, 5)
t=4	(0, 4)	(4, 0)	(0, -4)	(-4, 0)	(5, 9)	(9, 5)	(5, 1)	(1, 5)
t=5	(0, 5)	(5, 0)	(0, -5)	(-5, 0)	(5, 10)	(10, 5)	(5, 0)	(0, 5)
.....								

分析

经过不断的扩散，所有点最终一定能碰到。时间扩散t呈现出单调性，同时题目求“最早的时刻”，考虑二分答案求解。

考虑check函数。每次判断在二分时间mid时间内,能否成功形成连通块。判断连通块用并查集来做：

要是两点之间的曼哈顿距离（就是横纵坐标的差值和）不超过时间的2倍（因为两个点都能扩散，所以相对的扩散速度会增倍），那么就说明两个点能在一起然后就拿并查集连边，最后如果只有一个连通块就说明该时间合法，就向左区间去二分。

1. 如果所有点连通，所有点会有且只有一个根节点。如果存在两个以上根节点，表明没法实现全部连通。

2. 需要计算两点间的距离，如果距离大于能扩展出来的最大距离，显然是没法连通的。

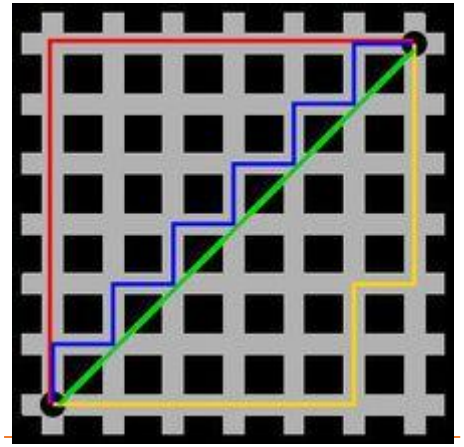
分析：数学相关

直线距离:A 点坐标为（x1, y1），B 点坐标为（x2, y2），那么 AB 之间的距离为 $|AB| = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ 。

曼哈顿距离

A点坐标为（x1, y1），B 点坐标为（x2, y2），那么 AB 之间的曼哈顿距离为 $AB = |x_1 - x_2| + |y_1 - y_2|$ 。

两个距离的关系



上图中红线代表曼哈顿距离，绿色代表欧氏距离，也就是直线距离，而蓝色和黄色代表等价的曼哈顿距离。

主函数（二分）

```
43  int main()
44  {
45      scanf("%d",&n);
46      for(int i=1;i<=n;i++)
47          scanf("%d%d",&a[i].x,&a[i].y);
48      for(int i=1;i<=n;i++)
49          for(int j=1;j<i;j++)//求两点间的曼哈顿距离
50              d[i][j]=d[j][i]=juli(a[i],a[j]);
51      int l=0,r=999999999,ans=0;
52      while(l<=r)
53      {
54          int mid=(l+r)/2;
55          if(check(mid)==true){r=mid-1;ans=mid;}
56          //目的求最小，正确往左移
57          else l=mid+1;
58      }
59      printf("%d\n",ans);
60      return 0;
61  }
```

主函数（定义）

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  struct node
4  {
5      int x,y;
6  }a[55];
7  int n,d[55][55];//d[i][j]记录距离,
8  int fa[55];//fa记录父亲
9
10 int juli(node n1,node n2){
11     //计算曼哈顿记录
12     return abs(n1.x-n2.x)+abs(n1.y-n2.y);
13 }
14
15 int find(int x)
16 {
17     if(x==fa[x])return x;
18     else return fa[x]=find(fa[x]);
19 }
```

主函数（check）

```
21  bool check(int x)//判断块是否连通
22  {
23      for(int i=1;i<=n;i++)fa[i]=i;//先规定好每个点的父亲节点
24      for(int i=1;i<=n;i++)//判断任意两点是否连通
25          for(int j=i+1;j<=n;j++)
26              if(d[i][j]<=x*2)//同样的距离小于两倍的时间
27              {
28                  int fx=find(i),fy=find(j);//寻找各自祖先
29                  if(fx!=fy)fa[fx]=fy;//如果根不同，合并
30              }
31
32      int sum=0;
33      for(int i=1;i<=n;i++)
34      {
35          if(fa[i]==i)sum++;//祖先相同的话，这样的走向就成立
36          if(sum==2)return false;//如果有两个根，表明不连通
37      }
38      return true;
39  }
```

[3637] 传送带

在一个2维平面上有两条传送带，每一条传送带可以看成是一条线段。两条传送带分别为线段AB和线段CD。lxh在AB上的移动速度为P，在CD上的移动速度为Q，在平面上的移动速度R。现在lxh想从A点走到D点，他想知道最少需要走多长时间

输入数据第一行是4个整数，表示A和B的坐标，分别为Ax，Ay，Bx，By；第二行是4个整数，表示C和D的坐标，分别为Cx，Cy，Dx，Dy；第三行是3个整数，分别是P，Q，R。

输出数据为一行，表示lxh从A点走到D点的最短时间，保留到小数点后2位。

输入样例

0 0 0 100

100 0 100 100

2 2 1

输出样例

136.60

分析

你要从 A 点到 D 点。有两条传送带：第一条从 A 到 B，速度为 p，第二条从 CC 到DD，速度为 q。不走传送带时速度为 r。求从 A 到 D 的最少时间。

明显这条路径应该是三条线段组成的：AE+EF+FD。其中 E 在 AB 上，F 在 CD 上。（以下设 dis(X,Y) 为 X 和 Y 的距离）

那么答案就是dis(A,E)/p+dis(E,F)/r+dis(F,D)/q。
这实际上是一个二元函数，想要求最小值还比较麻烦。
假设我们把其中一个元（这里取 E）当做参数会怎么样？
那么就是要找一点 F 使得 dis(E,F)/r+dis(F,D)/q 最小。
（以下设 f(X)=(dis(X,F)/r+dis(F,D)/q) min）
容易发现这是个单峰或者单调函数。

[3637] 传送带

f(E) 是可以通过三分 F 得到最小值的。

但是其实 E 也是个未知量啊！那怎么取得 dis(A,E)/p+f(E) 的最小值呢？

我们发现这也是个单峰或单调函数。可以再三分 EE 得出这个最小值，也就是答案。

综上：这题就是三分套三分。

[3637] 传送带

[1091] 自动刷题机

自动刷题机刷题的方式非常简单：首先会瞬间得出题目的正确做法，然后开始写程序。每秒，自动刷题机的代码生成模块会有两种可能的结果：

1. 写了 x 行代码
2. 心情不好，删掉了之前写的 y 行代码。（如果 y 大于当前代码长度则相当于全部删除。）

对于一个 OJ，存在某个固定的正整数长度 n，一旦自动刷题机在某秒结束时积累了大于等于 n行的代码，它就会自动提交并 AC 此题，然后新建一个文件（即弃置之前的所有代码）并开始写下一题。SHTSC 在某个 OJ 上跑了一天的自动刷题机，得到了很多条关于写代码的日志信息。他突然发现自己没有记录这个 OJ 的 n究竟是多少。所幸他通过自己在 OJ 上的 Rank 知道了自动刷题机一共切了 k 道题，希望你计算 n可能的最小值和最大值。

[1091] 自动刷题机

输入第一行两个整数 l , k，表示刷题机的日志一共有 l行，一共了切了 k 题。接下来 l 行，每行一个整数 xi，依次表示每条日志。若xi≥0，则表示写了xi 行代码，若xi<0，则表示删除了−xi 行代码。

输出一行两个整数，分别表示 n 可能的最小值和最大值。

如果这样的 n 不存在，请输出一行一个整数 −1。

样例输入

```
4 2
2
5
-3
9
```

样例输出

```
3 7
```

分析：

题目求满足某个条件的最大值和最小值。容易发现，对于给定的序列，n越大能过的题是越少的，符合单调性。所以可以采用二分答案，来求刚好过k道题的左右边界。

我们可以用两次二分来完成这道题，一次求最小值，一次求最大值。

check函数：

二分答案mid，要满足“自动刷题机一共成功了解决 k 道题”。因此check函数需要统计解题的数量。

注意题目的数据范围，xi<=1e9,1<=1e5,最大的sum值是1e14,数据要定义成 long long 类型

```
11 int check(LL u) {
12     LL cnt = 0, s = 0;
13     for (int i = 1; i <= n; i ++) {
14         s += a[i];//当前的累计值n
15         if (s < 0)
16             s = 0;
17         else if (s >= u){
18             s = 0;//重新计数
19             cnt ++ ;
20         }
21     }
22     return cnt;
23 }
```

```
 3  #include <bits/stdc++.h>
 4  using namespace std;
 5
 6  typedef long long LL;
 7  const int N = 100010;
 8  int n, k;
 9  int a[N];//每个时间的代码数量
10  LL l, r;
11  LL ans1, ans2;
12
13⊞ int check(LL u) {
26
27⊞ int main() {
```

```
13 int check(LL u) {
14     LL cnt = 0, s = 0;
15     for (int i = 1; i <= n; i ++) {
16         s += a[i];//当前的累计值n
17         if (s < 0)
18             s = 0;
19         else if (s >= u){
20             s = 0;//重新计数
21             cnt ++ ;
22         }
23     }
24     return cnt;
25 }
```

```
27 int main() {
28     scanf("%d%d", &n, &k);
29     for (int i = 1; i <= n; i ++)
30         scanf("%d", &a[i]);
31     ans1 = -1e18, ans2 = 1e18;
32     //ans1最大值    ans2 最小值
33     l = 1, r = 1e18;
34     while (l <= r) {//求最大值
35         LL mid = l + r >> 1;
36         if (check(mid) >= k)
37             ans1 = mid, l = mid + 1;
38         else
39             r = mid - 1;
40     }
```

```
42      l = 1, r = 1e18;
43      while (l <= r) {//求最小值
44          LL mid = l + r >> 1;
45          if (check(mid) <= k)
46              ans2 = mid, r = mid - 1;
47          else
48              l = mid + 1;
49      }
50
51      if (ans2 <= ans1)
52          printf("%lld %lld\n", ans2, ans1);
53      else
54          puts("-1");
55
56      return 0;
57  }
```

[8010] 牛牛大会

一场别开生面的牛吃草大会就要在 Farmer John 的农场举办了！世界各地的奶牛将会到达当地的机场，前来参会并且吃草。

具体地说，有 N 头奶牛到达了机场（$1 \le N \le 10^5$），其中奶牛 i 在时间 ti（$0 \le ti \le 10^9$）到达。Farmer John 安排了 M（$1 \le M \le 10^5$）辆大巴来机场接这些奶牛。每辆大巴可以乘坐 C 头奶牛（$1 \le C \le N$）。Farmer John 正在机场等待奶牛们到来，并且准备安排到达的奶牛们乘坐大巴。当最后一头乘坐某辆大巴的奶牛到达的时候，这辆大巴就可以发车了。Farmer John 想要做一个优秀的主办者，所以并不想让奶牛们在机场等待过长的时间。如果 Farmer John 合理地协调这些大巴，等待时间最长的奶牛等待的时间的最小值是多少？一头奶牛的等待时间等于她的到达时间与她乘坐的大巴的发车时间之差。

输入保证 $MC \ge N$。

[8010] 牛牛大会

输入的第一行包含三个空格分隔的整数 N（头牛），M（辆汽车），和 C（车的容量）。第二行包含 N 个空格分隔的整数，表示每头奶牛到达的时间。

输出一行，包含所有到达的奶牛中的最大等待时间的最小值。

6 3 2

1 1 10 14 4 3

4

分析

本题的n为1e5,需要考虑<=n*logn的时间复杂度。另外本题求等待的时间的最小值，因为时间是有明确边界的，可以考虑二分。用二分求奶牛中的最大等待时间的最小值。通过判定 按此时间划分奶牛所需要的车辆数是否大于M 来检查这种方案是否合法。

为了节省车，我们尽可能让每一辆车上更多的牛，只要不超载即可；如果此时等下一头牛上车会导致最先上车的奶牛等车时间超过 X，那么就让这辆车出发，并立即换下一辆车来等候。

样例分析

6 3 2 4
1 1 10 14 4 3

如果两头时间 1 到达的奶牛乘坐一辆巴士，时间 3 和时间 4 到达的奶牛乘坐乘坐第二辆，时间 10 和时间 14 到达的奶牛乘坐第三辆，那么等待时间最长的奶牛等待了 4 个单位时间（时间 10 到达的奶牛从时间10等到了时间 14）。

每头奶牛等待的时间
=当前牛到达的时间 - 该车第一头牛上车的时间
=a[i]-lst

a[i]:她的到达时间与她乘坐的大巴的发车时间之差

分析

check(mid)函数满足以下条件：

1. 尽可能让每一辆车上更多的牛，只要不超载即可。坐满，开始下一辆车；

2. 如果此时等下一头牛上车会导致最先上车的奶牛等车时间超过 X，那么就让这辆车出发，并立即换下一辆车来等候。

a[i]:她的到达时间与她乘坐的大巴的发车时间之差

分析

```
 9  bool check(int x)// x最大等待时间的最小值
10  {
11      int cnt=1,lst=a[1],cntcow=1;
12      //车的数量，第一只上车的奶牛到站的时间，目前车上奶牛只数
13      for(int i=2;i<=n;i++)/
14      {//因为我们默认装在第一头奶牛，所以i从2开始
15          cntcow++;//加一头
16          if(a[i]-lst>x||cntcow>c)
17          {//如果等待时间超过了二分中点mid
18           //或当前车辆已经满载，就重新发一辆车
19              cnt++;//发车
20              cntcow=1;
21              lst=a[i];//调下一辆车来
22              if(cnt>m) return 0;//如果超过车数，就返回0
23          }
24      }
25      return 1;//方法可行
26  }
```

分析

```
1   #include<cstdio>
2   #include<algorithm>
3   using namespace std;
4   int n,m,c,a[100010];
5   bool check(int x)// x最大等待时间的最小值
6   {
23  int main() {
24      scanf("%d%d%d",&n,&m,&c);
25      for(int i=1;i<=n;i++)scanf("%d",&a[i]);
26      sort(a+1,a+n+1);
27      int l=0,r=a[n]-a[1];
28      while(l<r) {
29          int mid=(l+r)>>1;
30          if(check(mid))r=mid;//该时间下车辆数充足，等待时间可以更小
31          else l=mid+1;////车辆不足，等待时间要更长
32      }
33      printf("%d",l);
34      return 0;
35  }
```

[4021] 01序列

我们称一个字符串为好字符串，指这个字符串中只包含0和1。现在有一个好字符串，求这个字符串中1恰好出现k次的子串有多少个。

输入第一行给出一个数字k，表示子串中1的个数。

第二行给出好字符串。

输出一个整数，表示好字符串中有多少个符合条件的子串

$0 \le k \le 10^6, |s| \le 10^6$

输入	输入
1	2
1010	01010
输出	输出
6	4

思路分析

子串的定义：将一个字符串从开头与结尾删去任意字符，形成新的字符串，称为原字符串的子串。

子序列的定义：将一个字符串删去任意字符，形成新的字符串，称为原字符串的子序列。

区别就是:子串在原串中是连续的，子序列不一定。

子串计数问题，一般是给定一个字符串，求满足条件的子串的个数。

思路分析-枚举

该题求满足“子串中1的个数>=1”的子串数量。

遍历该字符串中的每个字符(for(i=1;i<=n;i++))，不重不漏的计算出以该字符为开头的方案数。答案就是每个字符答案的累加。时间复杂度为$O(n^2)$

输入
1
1010
输出
6

1
10
101
1010
0
01
010
1
10
0

超时！

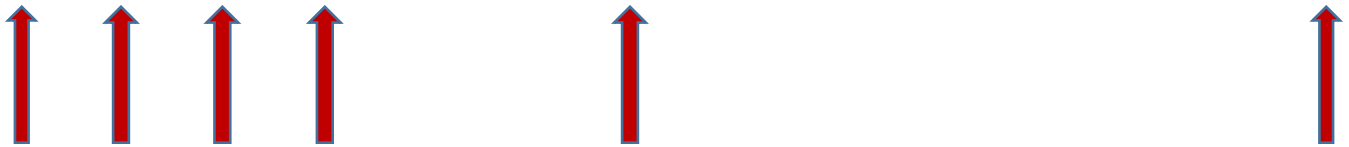
思路分析-枚举优化

在枚举子串时，是从左到右有序扫描的，有面向的左右区间和单调性，适合用二分优化。对于每个字符，我们记从当前字符开始，恰好出现k个1的最前位置为left，恰好出现k个1的最后位置为right，那么答案就是right-left+1，实际上，left和right是可以通过二分的到的。我们对每个字符都这样进行计算，即可得到答案。注意特判k=0。

复杂度:O(nlogn)

找到满足条件的左边界（>=k的最小值）

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
sum[i]	1	2	3	4	4	4	4	5	6	6	6	7	7	7
s[i]	1	1	1	1	0	0	0	1	1	0	0	1	0	0



i m m m m j

4
11110001100100

```
int l = i, r = n;
while (l < r) {
    int mid = l + r >> 1;
    if (isok())r = mid;
    else l = mid + 1;
}//找到满足条件的左边界
```

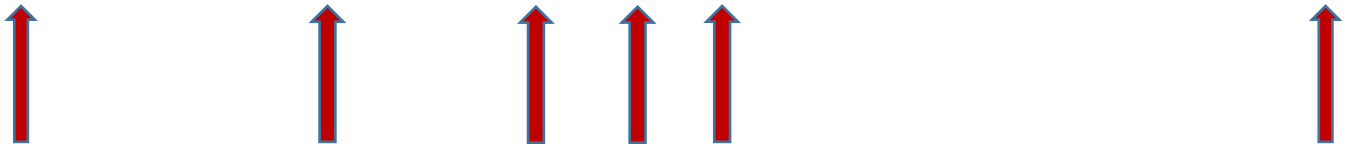
要下取整

```
for (int i = 1; i <= n; i++) {
    int l = i, r = n;
    while (l < r) {
        int mid = l + r >> 1;
        if (sum[mid] - sum[i - 1] >= k)r = mid;
        else l = mid + 1;
    }//找到满足条件的左边界

    if (sum[l] - sum[i-1] !=k)break;
      //如果左边界找不到说明不存在
    int pos = l;
```

找到满足条件的右边界（<=k的最大值）

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
sum[i]	1	2	3	4	4	4	4	5	6	6	6	7	7	7
s[i]	1	1	1	1	0	0	0	1	1	0	0	1	0	0



i m m m m j

```
l = i, r = n;
while (l < r) {
    int mid = l + r + 1 >> 1;
    if (isok())l = mid;
    else r = mid - 1;
}
```

m

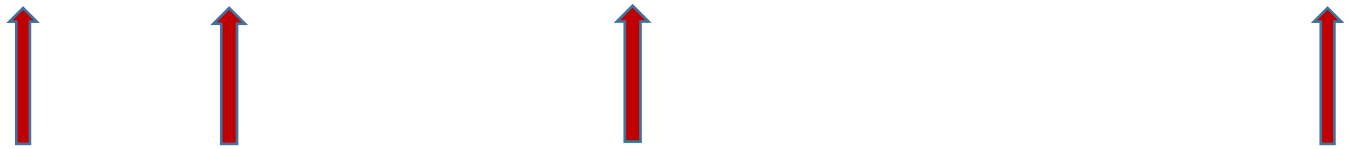
要上取整

```
    l = i, r = n;
    while (l < r) {
        int mid = l + r + 1 >> 1;//<=k
        if (sum[mid] - sum[i - 1] <= k)l = mid;
        else r = mid - 1;

    }
    ans += l - pos + 1;
}
```

找到满足条件的右边界（ ）

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
sum[i]	1	2	3	4	4	4	4	4	6	6	6	7	7	7
s[i]	1	1	1	1	0	0	0	0	1	0	0	1	0	0



i m m j

```
l = i, r = n;
while (l < r) {
    int mid = l + r + 1 >> 1;
    if (isok())l = mid;
    else r = mid - 1;
}
```

主函数代码

```
4   const int N = 1e6 + 10;
5   char s[N];//输入的原始数据
6   int sum[N];//存储1的前缀和
7   int k;//子串中包含1的个数
8   int main() {
9       scanf("%d", &k);
10      cin >> s + 1;
11      int n = strlen(s + 1);
12      if (k == 0) {
13          int pos = 0, ans = 0;
14          //特判k=0
15          for (int i = 1; i <= n; i++) {
16              if (s[i] == '0') {
17                  ++pos;
18              }
19              else {
20                  ans += (pos + 1) * pos / 2;
21                  pos = 0;
22              }
23          }
24          ans += (pos + 1) * pos / 2;
25          cout << ans;
26          return 0;
27      }
```

```
    for (int i = 1; i <= n; i++) {
        sum[i] = sum[i - 1] + (s[i] == '1');
    }//预处理前缀和
    int ans = 0;
    for (int i = 1; i <= n; i++) {
        int l = i, r = n;
        while (l < r) {
            int mid = l + r >> 1;
            if (sum[mid] - sum[i - 1] >= k)r = mid;
            else l = mid + 1;
        }//找到满足条件的左边界
        if (sum[l] - sum[i-1] !=k)break;//如果左边
        int pos = l;
        l = i, r = n;
        while (l < r) {
            int mid = l + r + 1 >> 1;
            if (sum[mid] - sum[i - 1] <= k)l = mid;
            else r = mid - 1;

        }
        ans += l - pos + 1;
    }
    cout << ans;
}
```

```
const int N = 1e6 + 10;
char s[N];//输入的原始数据
int sum[N];//存储1的前缀和
int k;//子串中包含1的个数
int main() {
    scanf("%d", &k);
    cin >> s + 1;
    int n = strlen(s + 1);
    if (k == 0) {
        int pos = 0, ans = 0;
        //特判k=0
        for (int i = 1; i <= n; i++) {
            if (s[i] == '0') {
                ++pos;
            }
            else {
                ans += (pos + 1) * pos / 2;
                pos = 0;
            }
        }
        ans += (pos + 1) * pos / 2;
        cout << ans;
        return 0;
    }
```

4. 实数二分

实数域上的二分，比整数二分简单。

```
1  const double eps =1e-7;//精度。
2  //如果下面用for，可以不要eps
3  while(right - left > eps){
4      //for(int i = 0; i<100; i++){
5          double mid = left+(right-left)/2;
6          if (check(mid))
7              right = mid; //判定，然后继续二分
8          else
9              left  = mid;
10 }
```

如果用for循环，由于循环内用了二分，执行100次，相当于实现了 1/2100的精度，一般比eps更精确

1.3 [1216] 陈晖的生日

陈晖的生日快到了，传统上晖妈要做披萨。晖妈会做各种口味，各种尺寸的披萨，数量为n个。陈晖的几个好朋友要来参加生日聚会，他们每人都得到一块馅饼（不是一个）。

小朋友们希望分到的披萨一样多，如果他们中的一个比其他人得到更大的一块，他们就会开始抱怨。因此，所有的小朋友都应该得到同样大小（但不一定形状相同）的馅饼，即使这会导致一些披萨变质（这比破坏聚会要好）。当然，陈晖自己也想要一块披萨，而且那块披萨的大小也应该一样。

请帮陈晖计算所有人能得到的最大尺寸是多少？所有的披萨都是圆柱形的，它们都有相同的高度1，但是披萨的半径可以不同。

1.3 [1216] 陈晖的生日

输入：第一行是一个正整数t(t<=100)：表示测试用例的数量。对于每一个测试数据；第二行两个整数N和F ，1≤ N、F≤ 10000：表示披萨的数量和朋友的数量。第三行有N个整数 ri，1≤ ri≤ 10000 表示每个披萨的半径。

输出：对于每个测试用例，输出一个最大的最大可能V，表示每小朋友人能分到的最大蛋糕是多大。答案是一个浮点数，绝对误差不超过10-3.

样例输入	样例输出
3 3 3 4 3 3 1 24 5 10 5 1 4 2 3 4 5 6 5 4 2	25.1327 3.1416 50.2655

分析

主人过生日，m个人来庆生，有n块半径不同的披萨，由m+1个人（加上主人）分，每人的披萨必须一样重，而且是一整块（不能是几个披萨碎块，也就是说，每个人的披萨都是从一块圆披萨中切下来的完整一块）。问每个人能分到的最大披萨是多大。

题解

最小值最大化问题。设每人能分到的披萨大小是x，用二分法枚举x。将每张披萨分成体积为x（x>=1）的F+1块。每张披萨可以有残留但不能使用多张披萨的残留凑成一小块。比如一个体积为5的披萨切出来了体积为4的一小块，剩余体积为1的不够再切一个4，直接丢弃。求x，误差须在0.001范围内。

二分求解。下界low=0，即每人都分不到披萨；上界high=maxsize，每人都得到整个披萨，而且那个披萨为所有披萨中最大的(上界就是 n个人n个披萨，每个披萨还等大)

对当前上下界折中为mid，计算"如果按照mid的尺寸分披萨，能分给多少人"；求某个披萨（尺寸为size）按照mid的尺寸，能够分给的人数，就直接size / mid，舍弃小数就可以。

代码

```
#include<stdio.h>
#include<math.h>
double PI = acos(-1.0);//3.141592653589793;
#define eps 1e-5
double area[10010];
int n,m;
bool check(double mid){
    int sum = 0;
    for(int i=0;i<n;i++) //把每个披萨都按大小mid分开。统计总数
        sum += (int)(area[i] / mid);
    if(sum >= m) return true;   //最后看总数够不够m个
    else         return false;
}
```

代码

```
14  int main(){
15      int T; scanf("%d",&T);
16      while(T--){
17          scanf("%d%d",&n,&m); m++;
18          double maxx = 0;
19          for(int i=0;i<n;i++){
20              int r; scanf("%d",&r);
21              area[i] = PI*r*r;
22              if(maxx < area[i]) maxx = area[i]; //最大的一块披萨
23          }
24          double left = 0, right = maxx;
25          for(int i = 0; i<100; i++){
26        //while((right-left) > eps)   {       //for或者while都行
27              double mid = left+(right-left)/2;
28              if(check(mid))    left  = mid;    //每人能分到mid大小的披萨
29              else              right = mid;    //不够分到mid大小的披萨
30          }
31          printf("%.4f\n",left);     // 打印right也对
32      }
33      return 0;
34  }
```

今天的课程结束啦……

下课了…
同学们再见！