二分搜索 2

在上一讲，我们学习了二分搜索的模板，利用二分查找的思想来优化算法，解决一类求最大值最小的问题。这一讲，我们继续基于二分搜索的模板，解决求最小值最大问题和对于实数区间进行二分的问题。

最小值最大

例：在单调递增序列 a 中查找<=x 的数中最大的一个

分析：
此题的模型就是求最小值最大。我们设定上下边界为 lb 和 ub，选定中间值 mid=(ub+lb)/2，对于元素 a[mid]，如果小于等于 x，则可以继续增大，令 lb=mid；如果元素 a[mid]大于 x，则需要变小，令 ub=mid。特别注意，在这里，我们定义的区间为前闭后开的，即[lb,ub)。

代码：

```
#include<iostream>
using namespace std;
int a[100],n,k;
int main(){
    cin>>n>>k;
    for(int i=1;i<=n;i++)cin>>a[i];
    int lb=0,ub=n+1;
    while(ub-lb>1){
        int mid=(ub+lb)/2;
        if(a[mid]<=k)lb=mid; //相当于区间[mid,ub)，a[mid]符合条件
        else ub=mid; //相当于区间[lb,mid)，已经确定 a[mid]不符合条件
    }
    cout<<lb;//最后剩下一个元素，即 lb
}
```

对于区间的定义，还可以定义闭合区间，即[lb,ub]，那么所对应的二分模板就需要改变了。二分区间的定义和模板的不同写法，是需要用心揣摩的，否则极易混乱写错。

```
#include<iostream>
using namespace std;
int a[100],n,k;
int main(){
    cin>>n>>k;
    for(int i=1;i<=n;i++)cin>>a[i];
    int lb=0,ub=n;
    while(lb<ub){
```

```
        int mid=(ub+lb)/2;
        if(a[mid]<=k)lb=mid; //相当于区间[mid,ub]，a[mid]符合条件
        else ub=mid-1; //相当于区间[lb,mid-1]，已经确定 a[mid]不符合条件
    }
    cout<<lb;//最后 ub=lb，输出 ub 和 lb 都行
}
```

牛栏

【题目描述】
FJ 新建了一个有 N(2≤N≤100,000)个畜栏的畜棚。畜栏的位置分布在直线的点 X_1，……X_N(0≤X_i≤1,000,000,000)上。
他有 C(2≤C≤N)只牛不喜欢这个畜棚的设计，并且对在同一个畜栏里的其他牛进行攻击。为了防止牛受到伤害， FJ 想把这些不听话的牛分配到某些畜栏中，使得这些牛所在的任意两个畜栏之间的最短距离尽可能长。求最长的最短距离是多少。
【输入】
第 1 行：两个用空格隔开的整数 N 和 C。
第 2 到 N+1 行：每行包括一个整数，表示畜栏的位置 X_i。
【输出】
一个整数：最长的最短距离。
【样例输入】
5 3
1 2 8 4 9
【样例输出】
3
【样例说明】FJ 把 3 只牛放到位置是 1、4 和 8 的畜栏里，最长的最短距离是 3。
【分析】
假设一个最短距离 x，验证以 x 为最短距离能否放置 c 头牛，如果可以，将 x 扩大再试，如果不可以，将 x 缩小再试。标准二分求最小值最大。
【代码】

```
#include<iostream>
#include<algorithm>
using namespace std;
int n,c,a[100001],ub,lb;
bool check(int x){    //验证以 x 为最短距离能否放置 c 头牛
    int sum=1,p=a[1];    //sum 是放置的牛数，p 为最后一头牛放置的位置
    for(int i=2;i<=n;i++){  //枚举每个牛棚
        if(a[i]-p>=x){  //如果这个牛棚与上一头牛的距离大于等于 x
            sum+=1;  //则这个牛棚可以放置牛，放置牛的数量加 1
            p=a[i];  //记录最后一头牛放置在 a[i]位置
        }
    }
    return sum>=c;    //如果可以安置 c 头牛，则 x 可行
```

```
}
int  main(){
        cin>>n>>c;
        for(int  i=1;i<=n;i++){
                cin>>a[i];
                ub=max(ub,a[i]);
        }
        sort(a+1,a+n+1);    //将牛棚排序
        while(ub-lb>1){
                int  mid=(ub+lb)/2;  //假定可行解
                if(check(mid))lb=mid;  //如果可行解可行，扩大区间
                else  ub=mid;    //否则缩小区间
        }
        cout<<lb;
}
```

跳石头

【题目描述】
一年一度的“跳石头”比赛又要开始了！
这项比赛将在一条笔直的河道中进行，河道中分布着一些巨大岩石。组委会已经选择好了两块岩石作为比赛起点和终点。在起点和终点之间，有 N 块岩石（不含起点和终点的岩石）。在比赛过程中，选手们将从起点出发，每一步跳向相邻的岩石，直至到达终点。
为了提高比赛难度，组委会计划移走一些岩石，使得选手们在比赛过程中的最短跳跃距离尽可能长。由于预算限制，组委会至多从起点和终点之间移走 M 块岩石（不能移走起点和终点的岩石）。
【输入】
输入文件第一行包含三个整数 L，N，M，分别表示起点到终点的距离，起点和终点之间的岩石数，以及组委会至多移走的岩石数。
接下来 N 行，每行一个整数，第 i 行的整数 Di（0 < Di < L）表示第 i 块岩石与起点的距离。这些岩石按与起点距离从小到大的顺序给出，且不会有两个岩石出现在同一个位置。
【输出】
输出文件只包含一个整数，即最短跳跃距离的最大值。
【样例输入】
25 5 2
2
11
14
17
21
【样例输出】
4
【提示】
对于 20%的数据，0 ≤ M ≤ N ≤ 10。

对于 50%的数据，0 ≤ M ≤ N ≤ 100。
对于 100%的数据，0 ≤ M ≤ N ≤ 50,000，1 ≤ L ≤ 1,000,000,000。
【分析】
首先要读懂题意。最短跳跃距离最长，即最小值最大。二分搜索解决。
【代码】

```
#include<iostream>
using namespace std;
int L,n,m,a[50002],lb,ub;
bool check(int x){    //难点在于如何验证
        int p=a[0],sum=0;  //p 表示当前位置，sum 表示需要移走的石头数
        for(int i=1;i<=n+1;i++){    //枚举每个石头
                if(a[i]-p<x){    //如果当前石头与上个位置在 x 以内
                        sum++;  //则这个石头可以移走，sum 加 1
                }else{
                        p=a[i];    //否则，跳到这个石头上
                }
        }
        return sum<=m;
}
int main(){
        cin>>L>>n>>m;
        for(int i=1;i<=n;i++)cin>>a[i];
        a[n+1]=L;
        ub=L+1;
        while(ub-lb>1){
                int mid=(ub+lb)/2;
                if(check(mid))lb=mid;
                else ub=mid;
        }
        cout<<lb;
}
```

山头狙击战

【题目描述】
Lucky 为了掩护大部队，单枪匹马同敌人周旋，后来被敌人包围在某山头……，不过不用怕，Lucky 携带了足够的弹药，完全可以将涌上来的敌人一个一个干掉。Lucky 是个神枪手，只要他的枪膛中有子弹，他就能将在他射程 m（用从敌人位置到山头的直线距离算）以内的一个敌人瞬间射杀，如果在射程内没有敌人，Lucky 会等待敌人靠近然后射击。但他发现了一个致命的失误：他携带的枪是单发枪，每射出一发子弹都必须花 k 秒钟的时间装子弹。而凶残的敌人才不会花时间等你换子弹呢。他们始终在以 1m/s 的速度接近山头。而如果在一个敌人到达山头时 Lucky 无法将他击毙，那么我们可怜的 Lucky 就将牺牲在敌人的刺刀下。现在 Lucky 向你求助：要保住自己的性命并且歼灭所有敌人，Lucky 最多只能用多少时间给枪

装上一发子弹？
说明：假设一开始 Lucky 的枪中就有一发子弹，并且一旦确定一个装弹时间，Lucky 始终会用这个时间完成子弹的装卸。希望你能帮助 Lucky 脱离险境。
【输入】
第一行有两个整数 n 和 m，（2≤n≤100,000； 1≤m≤10,000,000）n 代表敌人个数，m 代表 Lucky 的射程。
接下来有 n 行，每行一个整数 mi，（1≤mi≤10,000,000），代表每个敌人一开始相对山头的距离（单位为米）。
【输出】
仅有一个整数，代表 Lucky 的换弹时间（单位为秒）。
【样例输入】
6 100
236
120
120
120
120
120
【样例输出】
25
【分析】
首先，要看出这道题是求最小值最大的题型。如果换弹时间是 0，当然所有敌人都可以射杀，如果时间是 1 呢？如果时间是 2 呢？题目是求尽可能长的换弹时间，显然是最小值最大的题型。设定一个换弹时间，验证它，如果可以就将时间扩大再试，否则缩小再试。
【代码】

```
#include<iostream>
#include<algorithm>
using namespace std;
int n,m,a[100001],lb,ub;
bool check(int x){  //关键是这里，怎样验证 x 的换弹时间可以射杀所有敌人
    int t=0;  //全局时间
    for(int i=1;i<=n;i++){  //枚举每个敌人
        if(a[i]-t>m){  //如果敌人在射杀范围外
            t+=(a[i]-t-m);  //等待敌人走到射杀范围内，所需时间为 a[i]-t-m
        }
        if(a[i]-t<0)return false;  //如果敌人已经到达山顶了，无解
        t+=x;  //射杀一个敌人，开始换弹，需要 x 时间
    }
    return true;  //所有敌人射杀完，换弹时间可行
}
int main(){
    cin>>n>>m;
    for(int i=1;i<=n;i++){
        cin>>a[i];
```

```
    }
    sort(a+1,a+n+1); //将敌人按距离排序
    ub=a[n];  //换弹时间的理论上限
    while(ub-lb>1){  //最小值最大的模板
        int mid=(ub+lb)/2;
        if(check(mid))lb=mid;
        else ub=mid;
    }
    cout<<lb;
}
```

实数区间的二分

有时候问题的答案是在实数范围内的，那么我们也可以进行实数区间的二分，这个要精确到小数点后的位数。我们先看下模板：

```
while(ub-lb>1e-5){
    double mid=(ub+lb)/2;
    if(check(mid))ub=mid;
    else lb=mid;
}
```

在这个模板中，我们只需将循环的结束条件改为 ub-lb>1e-5，1e-5 表示 1 的负 5 次方，即 0.00001，即可对我们的答案精确到小数点后 5 位。当然，这里的区间定义还是半闭合区间，全闭合区间对待实数区间没法正常应对，这也是我们选择以半闭合区间为模板的原因。

消息传递

【题目描述】
有 N 个人在一直线上，第 i 个人的位置为 Di，满足 Di≤Di+1。最初只有第 1 个人(在最左边)知道消息。
在任意时刻，每个人可以以每秒 1 单位的速度向左或向右移动，或者停在原地。如果两个人的距离不超过 K，那么就可以进行消息传递。求所有人都知道消息最少需要多少时间。
【输入】
第一行一个正实数 K，表示最大的消息传递距离；
第二行一个整数 N，表示有 N 个人；
以下 N 行，每行一个正实数，表示每个人的位置，第 i 行表示第 i 个人的位置 Di。
【输出】
输出共一行一个实数，即所有人知道消息的最短时间。(结果保留三位小数点，四舍五入)
【样例输入】
3.000
2
0.000

6.000
【样例输出】
1.500
【数据规模】
对于 30%的数据：1≤N≤2,000；
对于 100%的数据：0≤K≤10^6；1≤N≤10^5；0≤Di≤10^9；
【分析】
这是在实数区间上求最大值最小。
【代码】

```
#include<cstdio>
#include<cstring>
#include<algorithm>
using namespace std;
int n;
double k,l=0,r=1e9,d[100010];
bool check(double x){  //验证以 x 能否传递完消息
    double left=d[1]+x;  //第一个人往右走的最远距离
    for(int i=2;i<=n;i++){
        if(left+k<d[i]-x)  //上一个人传输到的最远距离 left+k，d[i]-x 指 i 往做走
            return false;  //传送不到，退出
        left+=k; //传送得到，最远可以在 left+k 的位置接受信息
        if(d[i]+x<left)  //如果 left+k 比这个往右走还大，
            left=d[i]+x;  //则只能在 d[i]+x 的位置接受信息
    }
    return true;
}
int main(){
    scanf("%lf%d",&k,&n);
    for(int i=1;i<=n;i++)
        scanf("%lf",d+i);
    while(r-l>=1e-4){  //保留到小数点后 3 位，将区间精确到 0.0001
        double mid=(l+r)/2;  //最大值最小的模板
        if(check(mid))
            r=mid;
        else
            l=mid;
    }
    printf("%.3lf\n",r);
    return 0;
}
```

包裹快递

【题目描述】
一个快递公司要将 n 个包裹分别送到 n 个地方，并分配给邮递员 MCHacker 一个事先设定好的路线，MCHacker 需要开车按照路线给的地点顺序相继送达，且不能遗漏一个地点。MCHacker 得到每个地方可以签收的时间段，并且也知道路线中一个地方到下一个地方的距离。若到达某一个地方的时间早于可以签收的时间段，则必须在这个地方停留至可以签收，但不能晚于签收的时间段，可以认为签收的过程是瞬间完成的。
为了节省燃料，MCHacker 希望在全部送达的情况下，车的最大速度越小越好，就找到了你给他设计一种方案，并求出车的最大速度最小是多少。
【输入】
第 1 行为一个正整数 n，表示需要运送包裹的地点数。
下面 n 行，第 i+1 行有 3 个正整数 xi，yi，si，表示按路线顺序给出第 i 个地点签收包裹的时间段为[xi, yi]，即最早为距出发时刻 xi，最晚为距出发时刻 yi，从前一个地点到达第 i 个地点距离为 si，且保证路线中 xi 递增。
可以认为 s1 为出发的地方到第 1 个地点的距离，且出发时刻为 0。
【输出】
仅包括一个整数，为车的最大速度最小值，结果保留两位小数。
【样例输入】
3
1 2 2
6 6 2
7 8 4
【样例输出】
2.00
【限制】
对于 20%的数据，n≤10；
对于 30%的数据，xi，yi，si≤1000。
对于 50%的数据，n≤1000；
对于 100%的数据，n≤200000；xi≤yi≤108xi≤yi≤108；si≤107si≤107。
【提示】
第一段用 1 的速度在时间 2 到达第 1 个地点，第二段用 0.5 的速度在时间 6 到达第 2 个地点，第三段用 2 的速度在时间 8 到达第 3 个地点。
【分析】
与上题类似，本题同样是在实数区间求最大值最小。
【代码】

```
#include<iostream>
#include<iomanip>
using namespace std;
int n,x[200001],y[200001],s[200001];
long double lb=0,ub=99999999;
bool check(long double v){
    long double t=0;
    for(int i=1;i<=n;i++){
        t+=s[i]/v;  //到达 s[i]的时间
```

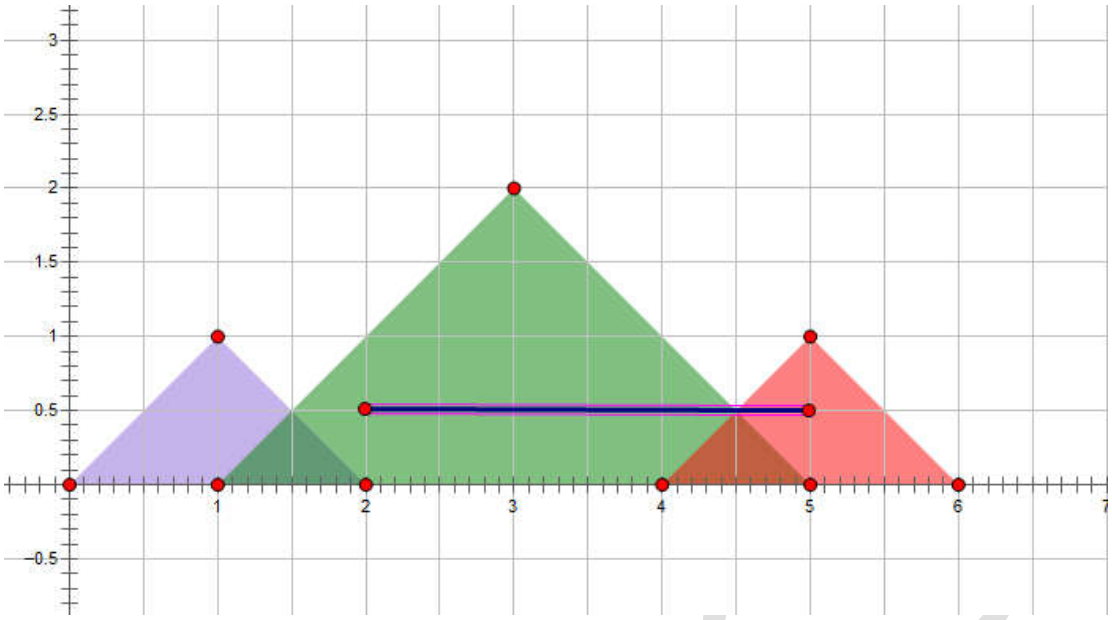
oiClass

```
        if(t<x[i])t=x[i];  //如果提前到达就等待
        if(t>y[i])return false;  //如果无法到达就返回 false
    }
    return true;
}
int main(){
    cin>>n;
    for(int i=1;i<=n;i++){
        cin>>x[i]>>y[i]>>s[i];
    }
    while(ub-lb>1e-5){
        long double mid=(ub+lb)/2;
        if(check(mid))ub=mid;
        else lb=mid;
    }
    cout<<setprecision(2)<<fixed<<ub;
}
```

开车出行

【题目描述】
在 2117 年*国为了缓解交通压力开发了空中汽车，为了保证夜间行驶的安全在各个路段安装了一系列路灯进行照明。已知路灯的照明范围是一个以路灯所在的点为顶点的道路最低的直线为底边的等腰直角三角形。汽车的行驶区域必须有路灯的照明。我们为了简化问题将道路简化为一条直线。建立平面直角坐标系，并定义道路的最低点所在的直线为 x 轴。给定所有路灯在直角坐标系的位置以及一个车辆需要从点 a 行驶到点 b。问这辆汽车所能行驶的最高高度。
【输入】
第一行一个整数 n 表示路灯的数量
接下来 n 行每行两个整数(x,y)表示第 i 个路灯所在的位置。
第 n+2 行两个整数 x1,x2 表示 a，b 两点的 x 坐标位置
输入数据保证 n<=100000 所有的坐标均为整数且绝对值小于等于 10000
【输出】
一个数表示能够行驶的最高高度
精确到小数点后三位小数。
本题采用实数比较的方式进行答案正确性判定
【样例输入】
3
1 1
3 2
5 1
2 5
【样例输出】
0.500

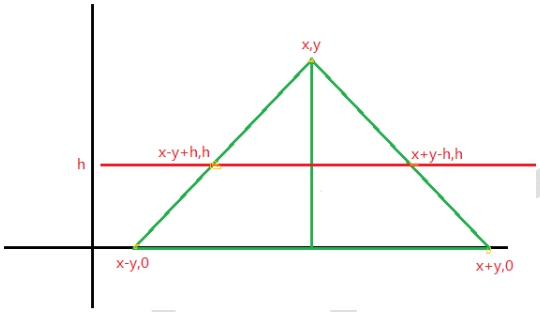
【样例解释】



图中的三角形分别表示了路灯的照射范围，可以看出由 2 到 5 最高的高度为 0.5。

【分析】

本题是在实数区间求最小值最大。需要特别注意的点是对于等腰直角三角形，其顶点为(x,y)，那么它的左端点为（x-y,0），右端点为（x+y,0），如果有一条高为 h 的线穿过三角形，则线与三角形的左右交点分别为（x-y+h,h），(x+y-h,h)，如果不理解这个，可以自己画图证明一下。知道这个条件后，就不难实现这个问题的求解了，详见代码。



【代码】

```
#include<iostream>
#include<algorithm>
#include<iomanip>
using namespace std;
struct point{
    int x,y;
}light[100001];
int n,a,b;
double lb,ub=10000;
bool cmp(point a,point b){
    return a.x-a.y<b.x-b.y;  //x-y 是左端点的位置
}
```

```
bool check(double x){  //判断 x 的高度能够被所有三角形覆盖到
    double p=a;  //一开始从 a 为起点
    for(int i=1;i<=n;i++){
        double lt=light[i].x-light[i].y;  //lt 表示第 i 个三角形的左端点
        double rt=light[i].x+light[i].y;  //rt 表示第 i 个三角形的右端点
        if(p<lt+x)return false;  //如果 p 不能与左边界连接，无解
        if(rt-x>p)p=rt-x;  //可以连接到右边界
        if(p>=b)return true;  //如果 p 超过终点 b，则方案可行
    }
}
int main(){
    cin>>n;
    for(int i=1;i<=n;i++)cin>>light[i].x>>light[i].y;
    cin>>a>>b;
    sort(light+1,light+n+1,cmp);  //按照三角形的左端点进行排序
    while(ub-lb>1e-5){  //最小值最大的模板
        double mid=(ub+lb)/2;
        if(check(mid))lb=mid;
        else ub=mid;
    }
    cout<<setprecision(3)<<fixed<<lb;
}
```