# 二分答案

```
枚举答案:
for(i:遍历答案区间){
    if(答案满足条件)
        保存答案或统计个数
}
```

顺序遍历答案区间，当区间非常大的时候会超时
将二分的思想引入到枚举答案中
加速答案的枚举，就成为了二分答案


信息学

# 二分答案

# 二分答案的前提


**二分答案的前提：**

- 答案范围有已知的上、下界[L,R]
- 具有单调性(递增、递减)

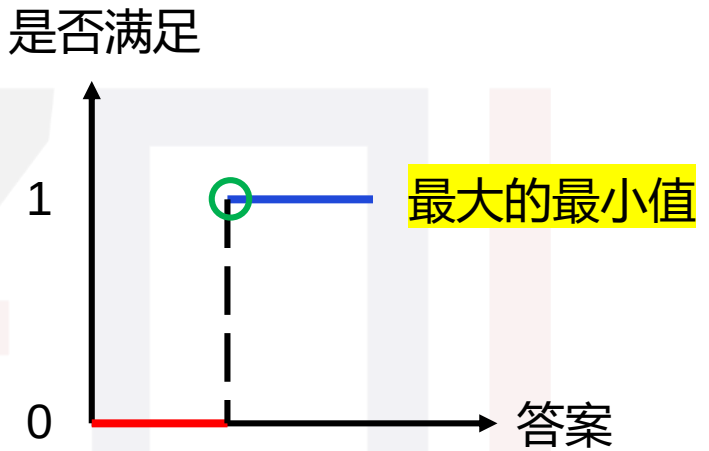
例如：答案区间为1 2 3 4 5 6 8 9 17，则为递增；反过来就是递减

# 二分答案的正确性

假设：这是递增的答案区间

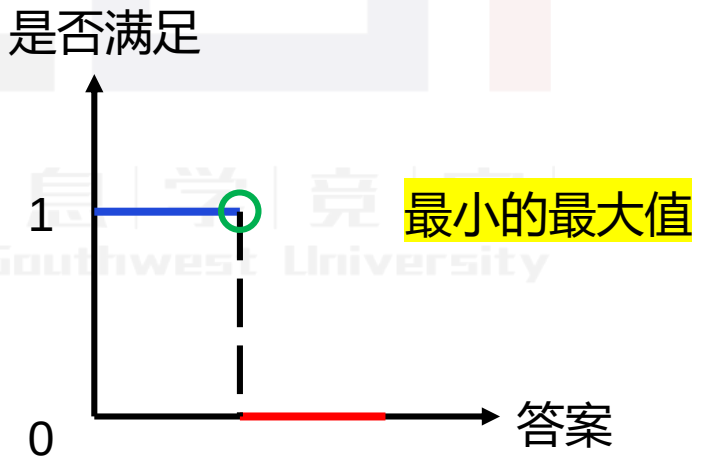
第一种情况：

答案在后半段


第二种情况：

答案在前半段

# 二分答案适用范围

1. 求最大的最小值（NOIP2015跳石头）。
2. 求最小的最大值（NOIP2010关押罪犯）。
3. 求满足条件下的最小（大）值。
4. 求最靠近一个值的值。
5. 求最小的能满足条件的代价

这种问题常常有关键语句：使最大……最小,使最大……最大、……至少是多少、求出最少的……

# 二分答案算法模板(整数情况下近似万能)

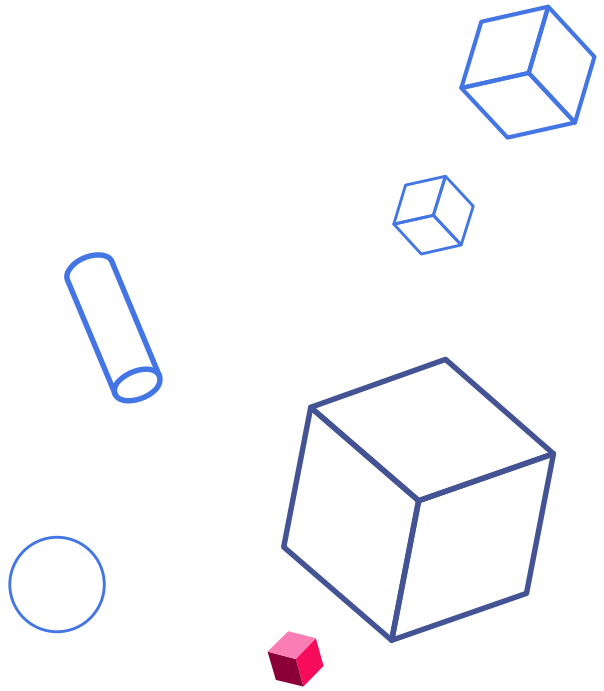
```
int left = low, right = high, ans = 0;
while(left <= right){
    int mid = (left + right) >> 1;  //移位
    if( check(mid) ) { //check函数是二分答案的关键
       ans = mid;
       left = mid + 1;
    }
      else right = mid - 1;
 }
```

最后结果保存在ans里面

程序说明：

- [left,right]：答案可能的上、下界low,high
- mid：二分枚举的答案
- check()检查mid的正确性：返回1，合法；返回0，不合法

# 例题讲解



# 例1：木材加工

木材厂有一些原木，现在想把这些木头切割成一些长度相同的小段木头（木头有可能有剩余），需要得到的小段的数目是给定的。当然，我们希望得到的小段木头越长越好，你的任务是计算能够得到的小段木头的最大长度。木头长度的单位是cm。原木的长度都是正整数，我们要求切割得到的小段木头的长度也是正整数。
例如有两根原木长度分别为11和21，要求切割成到等长的6段，很明显能切割出来的小段木头长度最长为5。

输入:
第一行是两个正整数N和K(1 ≤ N ≤ 100000，1 ≤ K ≤ 100000000)，N是原木的数目，K是需要得到的小段的数目。
接下来的N行，每行有一个1到100000000之间的正整数，表示一根原木的长度。
输出:
能够切割得到的小段的最大长度。如果连1cm长的小段都切不出来，输出”0”。

样例输入：
3 7
232
124
456

样例输出：
114

# 思路

猜木头长度，二分木头长度

**Q：答案的范围**

1~maxn，maxn为最长原木的长度

**Q：什么样的长度是合法的?**

每段木头长度为m时，能锯的总段数sum>=k;

当长度为m时，若sum>=k，说明还可以再锯得长一点，left+1

当长度为m时，若sum<k，说明长度太长了，right-1

# check()函数

作用：检查长度为m时，是否能切不少于k段木头

```
int check(int m)
{
    sum=0;  //sum统计能切多少段
    for(int i=1;i<=n;i++) //枚举每段原木
       sum+=a[i]/m; //累加木头段数
    if(sum>=k) return 1; //如果能够满足k段木头，返回1
    else return 0; //不够k段木头，返回0
}
```

# 二分答案核心代码

```
left=1,right=maxn,ans=0;
while(left<=right){ //二分模板
    mid=(left+right)/2;
    if(check(mid)){
        ans=mid;
        left=mid+1;
    }
    else right=mid-1;
}
```

# 例2：跳石头

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）。

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。
输出只包含一个整数，即最短跳跃距离的最大值

| 样例输入 | 样例输出 |
| --- | --- |
| 25 5 2<br>2<br>11<br>14<br>17<br>21 | 4 |

# 思路

## 二分距离

**Q：距离的范围?**

最小值为0，最大值为终点的距离

**Q：什么样的石头应该挪走?**

对于一对石头a[i]与a[j]，它们之间的距离**(a[j]-a[i])>=所猜的数mid**，
就把它们中间的石头移除，因为这样才能扩大他们之间的距离。

若距离mid满足条件，继续在[mid+1,right]的范围枚举，left=mid+1

否则，继续在[left,mid-1]的范围枚举，right=mid-1

# check()函数

```
int check(int dis)
{
    int cnt=0,start=0;  //cnt记录需要移走的石头数,start标记起跳的石头
    for(int i=1;i<=n+1;i++){
      if(a[i]-start<dis) cnt++; //a[i]与起跳点的距离小于dis,挪走
      else start=a[i]; //大于dis,起跳点更新
    }
    if(cnt<=m) return 1; //若枚举的距离dis满足最多挪走m块石头，说明这个距离合法
    else return 0;   //否则这个距离不合法
}
```

# 二分答案核心代码

```
a[n+1]=L;  //a[n+1]存储终点的距离
left=0,right=L;
while(left<=right)  //二分答案模板
{
   mid=(left+right)/2;
   if(check(mid)){
      ans=mid;
      left=mid+1;
   }
   else right=mid-1;
}
```

# 总结

二分解题情境：最大的最小、最小的最大等，也可以和其他的算法结合使用

二分解题步骤：确定答案可能的范围，根据题意书写check函数(关键/难点)

# 拓展：小数的二分答案

前置知识：需要先行了解什么是根、实根

有形如：ax^3+bx^2+c^x+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：令f（x）=ax^3+bx^2+cx+d，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。

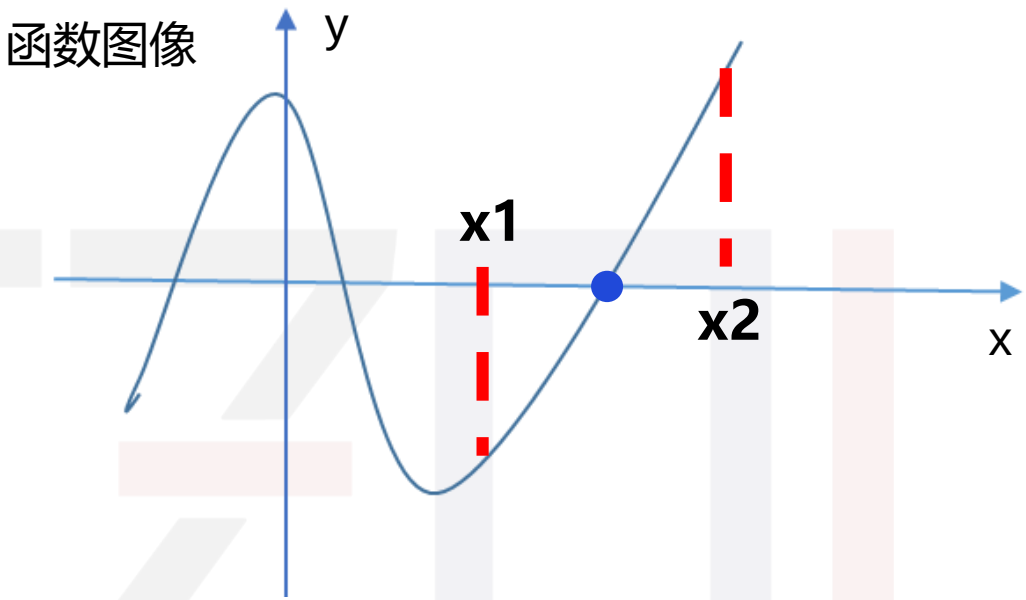
| 样例输入 | 样例输出 |
| --- | --- |
| **1 -5 -4 20** | **-2.00 2.00 5.00** |

**题意**　通过二分答案找到一元三次方程的三个实根

# 分析

**根据题目已知：**

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。




**1.二分什么样的答案?** 方程的根

**2.二分的范围?** [-100,100]?

在这个区间存在多个根，二分答案只能找到一个根

**根据题目条件：**

答案范围：[-100,100],并且两个根的差绝对值>=1. ➡ 保证了每一个大小为1的区间里至多有1个解

那么我们可以把[-100,100]这个区间**分成200个区间长度为1**，且互不重叠的小区间，在这个小区间至多只有一个根，我们可以在这个**小区间进行二分答案。**

# 参考代码

```
for(i=-100,i<=100;i++){  //枚举小区间
        x1=i,x2=i+1;  //每个小区间长度为1
        if(f(x1)==0)   printf("%.2lf",x1); //x1为根直接输出
        else if(f(x1)*f(x2)<0){    //存在根，进行二分
          while(x2-x1>=0.001){ //double精度问题，题目要求精确到后两位
               mid=(x1+x2)/2;
               if(f(x1)*f(mid)<=0) //存在根
                 x2=mid; //调整上边界
               else
                 x1=mid; //调整下边界
             }
             printf("%.2lf",x1); //输出根
         }
}
```

f()函数计算一元三次方程的值：

```
double f(double x)
{
    return x*x*x+b*1.00/a*x*x+c*1.00/a*x+d*1.00/a;
}
```

# Thanks

## For Your Watching