算法-二分

信智学院　陈琰宏

1.1 [1108] 查询元素

有一正整数序列a1,a2,a3,...an, 其中 $n<=10^5$ ， $ai<=10^9$ 有m 个查询（$m<=10^5$）,每次查询m是否在数组a中如果存在输出1，否则输出0.

样例输入

10 4
16 63 20 30 70 76 73 79 23 40
20
22
40
45

样例输出

1
0
1
0

方法1：直接模拟

顺序查找，执行m次。时间复杂度为0(m*n)

```
6       int m,n,x;
7       cin>>n>>m;
8       for(int i=1;i<=n;i++)
9           cin>>a[i];
10      for(int i=1;i<=m;i++){
11          cin>>x;
12          bool exist=false;
13          for(int j=1;j<=n;j++){
14              if(a[j]==x){
15                  exist=true;
16                  break;
17              }
18          }
19          if(exist)
20              printf("1\n");
21          else
22              printf("0\n");
23      }
```

$0(m*n)=10^{10}$

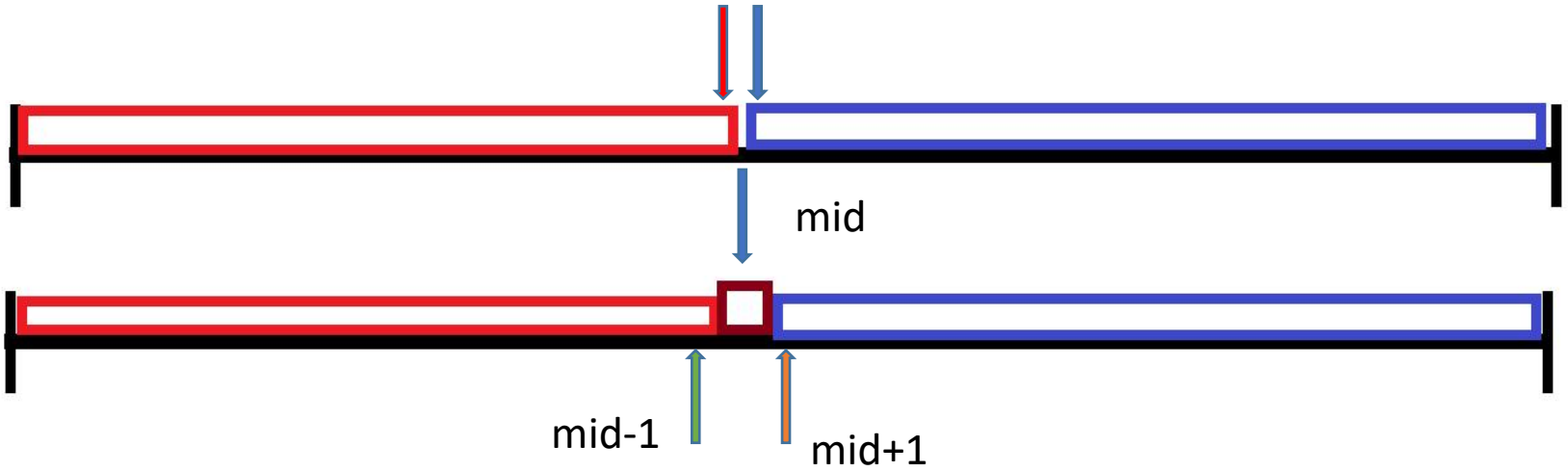
方法2：二分

二分法是一种在有序数组中查找某一特定元素的搜索算法。搜索过程从数组的中间元素开始，如果中间元素正好是要查找的元素，则搜索过程结束；如果某一特定元素大于或者小于中间元素，则在数组大于或小于中间元素的那一半中查找，而且跟开始一样从中间元素开始比较。如果在某一步骤数组为空，则代表找不到。这种搜索算法每一次比较都使搜索范围缩小一半。

二分查找是效率很高，时间复杂度为$\log_2 n$

二分

二分把一个区间一分为二，一个区间满足要求，一个区间不满足。二分就是寻找这个要求的边界。



什么情况下用二分？

上下界[a, b]确定、函数在[a, b]内单调

二分思路

1.对原数组排序，时间复杂度为 0(n*logn)

2. 查找x时限于中间的元素比较，等于则结束。中间数小于x则在右侧查找（调整左端点）；中间数大于x则在左侧查找（调整右端点），重复上述方法。时间复杂度为0(logn)。执行m次，时间复杂度为m*logn。

总的时间复杂度为0(n*logn+m*logn)

1.1 [1108]: 查询元素

给出n个不相同的正整数a1, a2, a3, . . . an, 其中 n<=10^5 , ai<=10^9

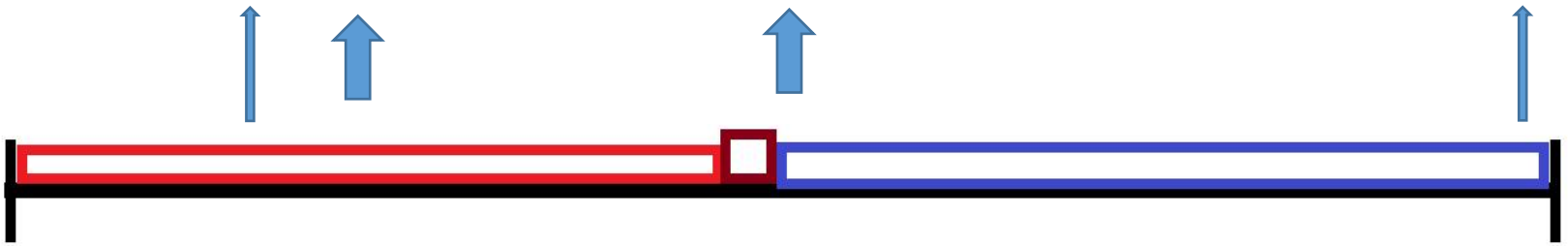
有m 个查询（m<=105）, 每次查询m是否在数组a中如果存在输出1，否则输出0.

序号	1	2	3	4	5	6	7	8	9	10
原数组	16	63	20	30	70	76	73	79	23	40

序号	1	2	3	4	5	6	7	8	9	10
原数组	16	20	23	30	40	63	70	73	76	79

1.1 [1108]: 查询元素

原数组	16	20	23	30	40	63	70	73	76	79



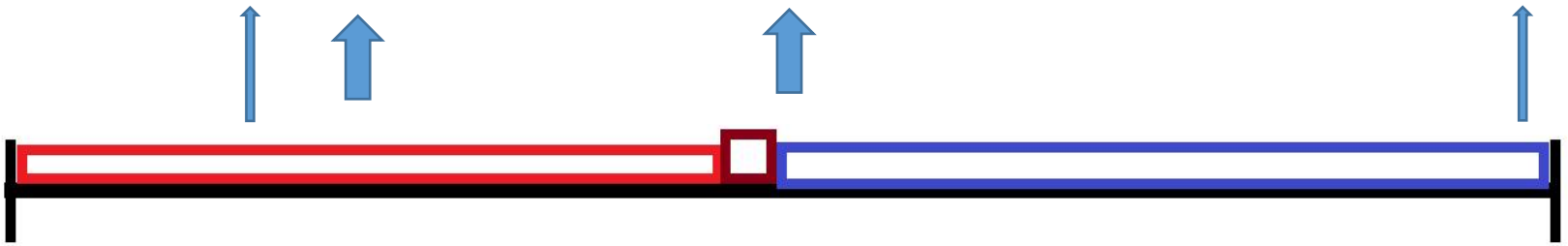
r=mid-1

l=mid+1

L	Mid	R	A[mid]		x
1	5	10	40	>	23
1	2	4	20	<	23
3	3	4	23	=	23

1.1 [1108]: 查询元素

原数组	16	20	23	30	40	63	70	73	76	79



r=mid-1

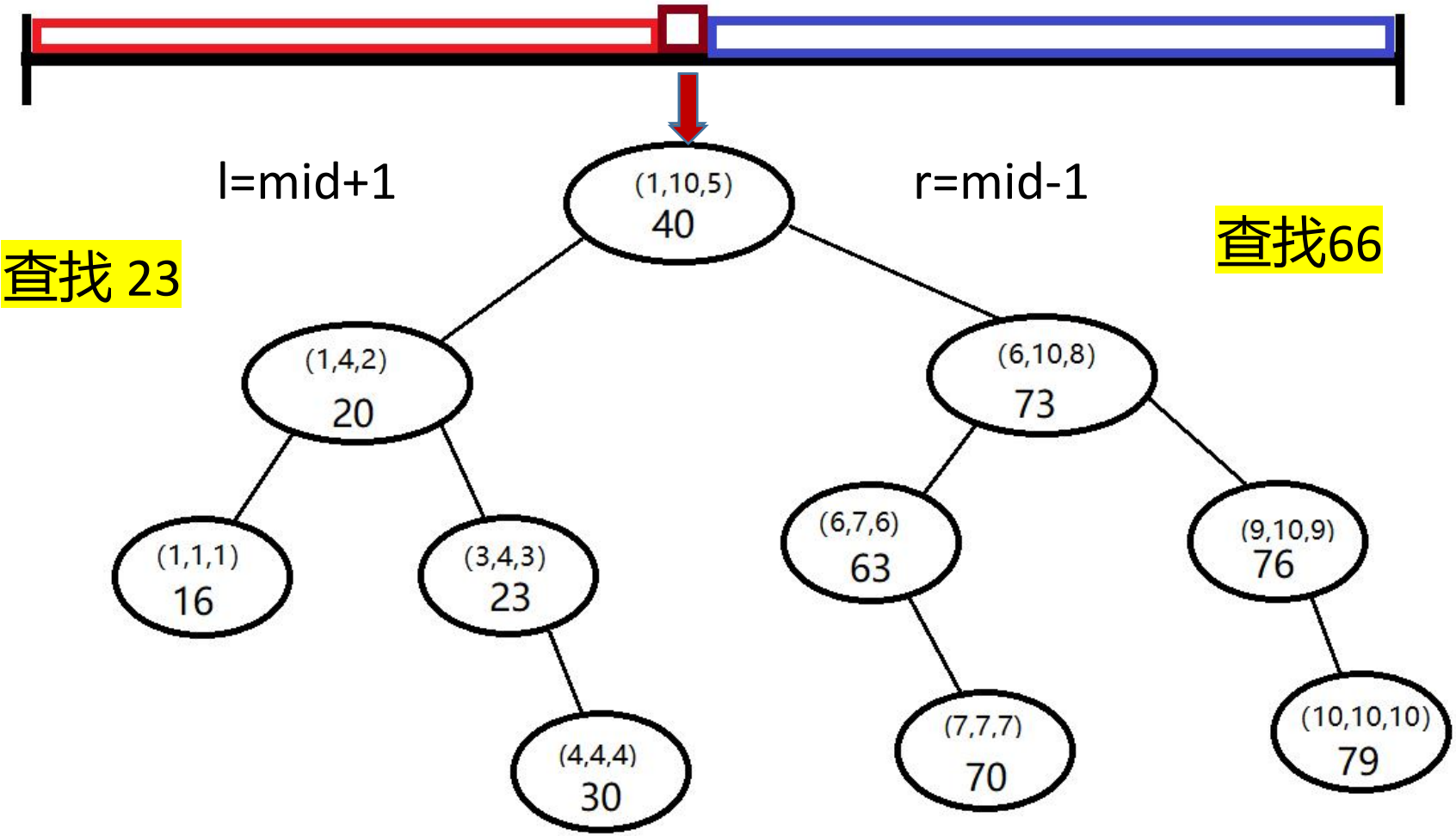
l=mid+1

L	Mid	R	A[mid]		x
1	5	10	40	>	22
1	2	4	20	<	22
3	3	4	23	>	22
3		2			

L>R

1.1 [1108]: 查询元素

原数组	16	20	23	30	40	63	70	73	76	79



1.1 [1108]: 查询元素

```
1  bool check(int a[],int x){
2      bool flag=false;
3      int l=1,r=n;
4      while(l<=r){
5          int mid=(l+r)>>1;
6          if(a[mid]==x){
7              flag=true;
8              break;
9          }
10         if(a[mid]<x)
11             l=mid+1;
12         else
13             r=mid-1;
14     }
15     return flag;
16 }
```

1.1 [1108]: 查询元素

```
1  //1 满足等于x的位置
2  int check(int a[],int x){
3
4      int l=1,r=n;
5      while(l<=r){
6          int mid=(l+r)>>1;
7          if(a[mid]<=x)
8              l=mid+1,ans=mid;
9          else//>= 右侧可能是答案
10             r=mid-1;
11     }
12     return ans;
13 }
```

通用写法!

1.2 [1154] 查询元素2

给出n个有序的正整数a1,a2,a3,...an, 其中 $n<=10^5$ ， $ai<=10^9$ 有m 个查询（$m<=10^5$）,每次查询输出刚好大于等于x的位置，如果不存在大于x的元素输出n+1。

输入

```
10  4
16  20  20  40  40  63  70  73  76  79
20
100
65
5
```

输出

```
2
11
7
1
```

分析

序号	1	2	3	4	5	6	7	8	9	10
数组	16	20	20	40	40	63	70	73	76	79

a数组有序，可以利用二分法实现查找，注意不存在大于等于x的情况可以单独特判。

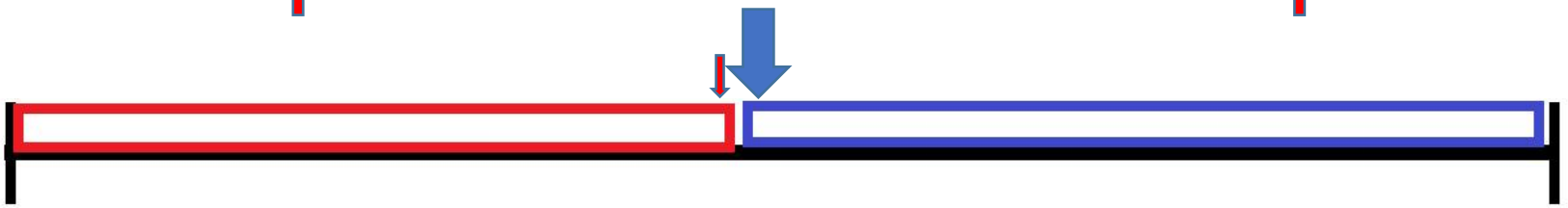
a[mid]<x时，mid不可能为答案，所以l=mid+1;

否则,mid可能为答案，所以r=mid;

此种情况注意 mid=(l+r)>>1;while（l<r）,l,r点为答案

分析：刚好大于等于x的位置(x>=k)

序号	1	2	3	4	5	6	7	8	9	10
数组	16	20	20	40	40	45	45	45	76	79



L	Mid	R	A[mid]		x		
1	5	10	40	<	45	该位置不符合要求舍弃	l=mid+1
6	8	10	45	≥	45	该位置符合要求	r=mid
6	7	8	45	≥	45	该位置符合要求	r=mid
6	6	7	45	≥	45	该位置符合要求	r=mid
6		6				L==R 找到位置 i=6，结束！	

1.2 满足≥x 最靠左（小）的位置

```
1  int check(int a[],int x){
2      if(a[n]<x)return n+1;
3      int l=1,r=n;
4      while(l<r){
5          int mid=(l+r)>>1;
6          if(a[mid]<x)//mid不可能是答案
7              l=mid+1;//左侧不可能是答案
8          else//>= 右侧可能是答案
9              r=mid;
10     }
11     return l;
12 }
```

1.3[1036] 数的范围

给定一个按照升序排列的长度为 n 的整数数组，以及 q 个查询。对于每个查询，返回一个元素k 的起始位置和终止位置（位置从 0 开始计数）。如果数组中不存在该元素，则返回 -1 -1。

输入第一行包含整数 n 和 q，表示数组长度和询问个数。第二行包含 n 个整数，表示完整数组。

接下来 q 行，每行包含一个整数 k，表示一个询问元素。

1≤n≤1000001　1 ≤q≤10000　　1≤k≤10000

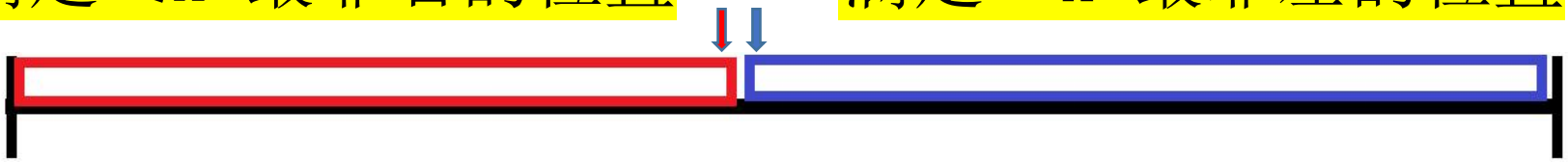
输出共 q 行，每行包含两个整数，表示所求元素的起始位置和终止位置。

如果数组中不存在该元素，则返回 -1 -1。

1.3 [1036] 数的范围

输入样例	输出样例
6 3	3 4
1 2 2 3 3 4	5 5
3	-1 -1
4	
5	

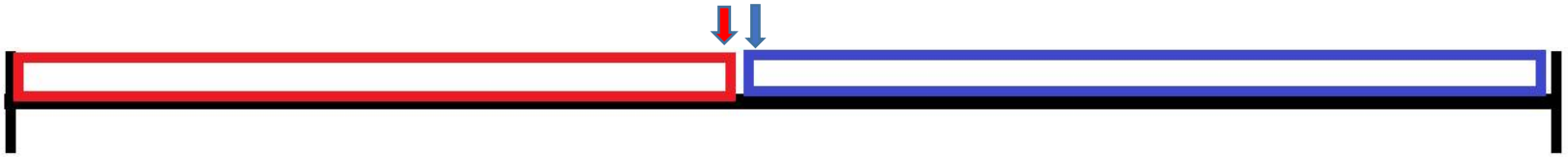
满足≤x 最靠右的位置　　满足 ≥x 最靠左的位置



1.3 满足≤x 最靠右（大）的位置

序号	1	2	3	4	5	6	7	8	9	10
数组	3	9	11	13	22	27	33	57	66	77

小于等于26的最大值

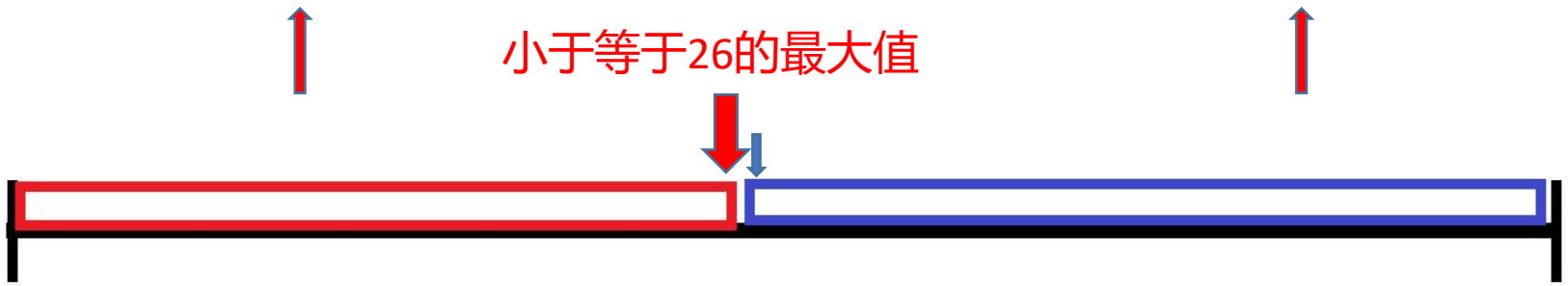


L	Mid=(l+r)/2	R	A[mid]		x		
1	5	10	22	≤	26	该位置符合要求	l=mid
5	7	10	33	>	26	该位置不符合要求	r=mid-1
5	5	6	22	≤	26	该位置符合要求	l=mid
5	5	6	22	≤	26	该位置符合要求	l=mid

陷入死循环！

1.3 满足≤x 最靠右（大）的位置

序号	1	2	3	4	5	6	7	8	9	10
数组	3	9	11	13	22	27	33	57	66	77



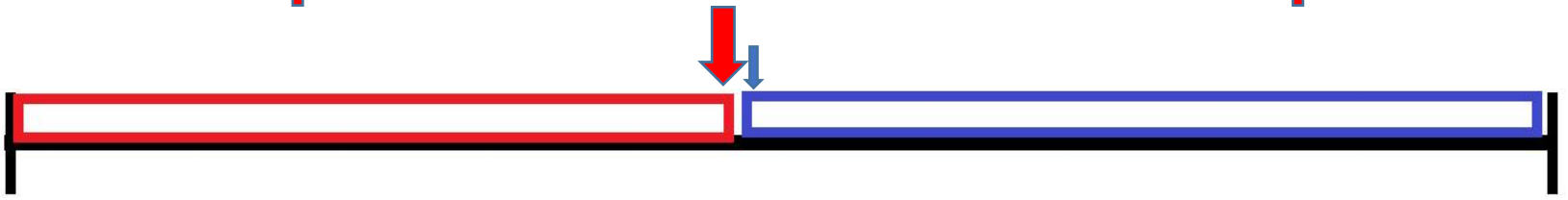
如何解决“满足要求最大值”的死循环问题？

上取整！mid=(l+r+1)/2

1.3 满足≤x 最靠右（大）的位置

序号	1	2	3	4	5	6	7	8	9	10
数组	3	9	11	13	22	27	33	57	66	77

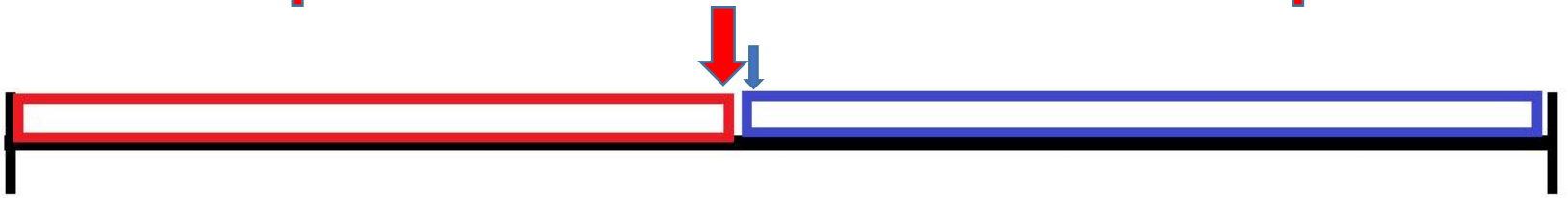
小于等于26的最大值



L	Mid=(l+r+1)/2	R	A[mid]		x		
1	6	10	27	>	26	该位置不符合要求	r=mid-1
1	3	5	11	≤	26	该位置符合要求	l=mid
3	4	5	13	≤	26	该位置不符合要求	l=mid
4	5	5	22	≤	26	该位置不符合要求	r=mid
5		5				l==r 得到结果 22对应的i，结束	

1.3 满足≤x 最靠右（大）的位置

序号	1	2	3	4	5	6	7	8	9	10
数组	16	20	20	40	40	45	45	45	76	79



L　Mid=(l+r+1)/2　R　A[mid]　x

1	6	10	45	≤	45	该位置符合要求　l=mid
6	8	10	45	≤	45	该位置符合要求　l=mid
8	9	10	76	≥	45	该位置不符合要求　r=mid-1
8		8				

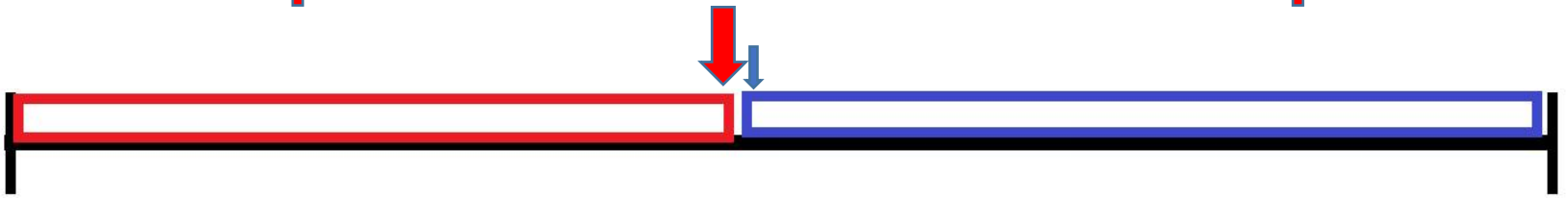
l==r 得到结果 对应的i=8，结束

```
1  //3 满足小于等于x 最靠右的位置
2  int check(int a[],int x){
3      if(a[n]<x)return n+1;
4      int l=1,r=n;
5      while(l<r){
6          int mid=(l+r+1)>>1;//取中心靠近右端点
7          if(a[mid]<=x)
8              l=mid;//左侧可能是答案
9          else//右侧不可能是答案
10             r=mid-1;
11     }
12     return l;
13 }
```

1.3 满足≤x 最靠右（大）的位置

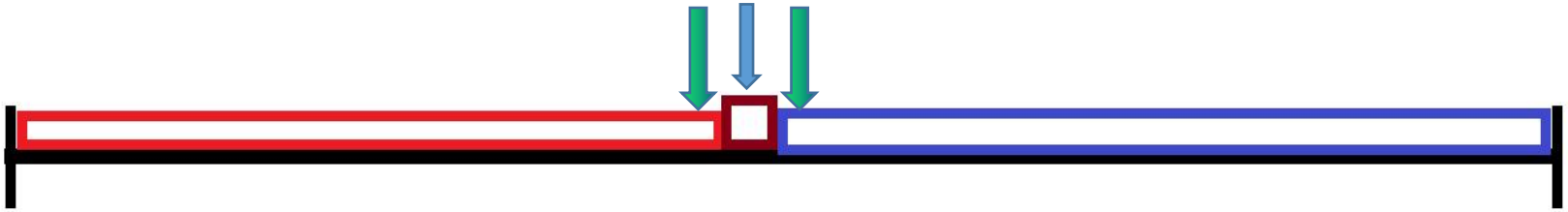
序号	1	2	3	4	5	6	7	8	9	10
数组	3	9	11	13	22	27	33	57	66	77

小于等于26的最大值



L	Mid=(l+r)/2	R	A[mid]		x		
1	6	10	27	>	26	该位置不符合要求	r=mid-1
1	3	5	11	≤	26	该位置符合要求	l=mid
3	4	5	13	≤	26	该位置不符合要求	l=mid
4	5	5	22	≤	26	该位置不符合要求	r=mid
5		5				l==r 得到结果 22对应的i，结束	

总结：二分的三种情况



1. 求满足等于x的位置

```
while(l<=r){ //check(mid)
mid=(l+r)/2;if(x<=a[mid]){lans=mid,r=mid-1} else {l=mid+1;} }
```

2. 求满足大于等于x的最左位置

```
while(l<r) {
    mid=(l+r)/2;//不符合要求        符合要求
    if(a[mid]<x){l=mid+1; } else    {r=mid;}
}
```

3. 求满足小于等于x的最右位置

```
while(l<r) {
    mid=(l+r+1)/2;//不符合要求        符合要求
    if(a[mid]>x){r=mid-1; } else    {l=mid;}
}
```

二分的三种情况

```
1  //1 满足等于x 的位置
2  bool check(int a[],int x){
3      bool flag=false;
4      int l=1,r=n;//设置左右端点
5      while(l<=r){
6          int mid=(l+r)>>1;
7          if(a[mid]==x){
8              flag=true;
9              break;
10         }//调整左右端点
11         if(a[mid]<x)
12             l=mid+1;
13         else
14             r=mid-1;
15     }
16     return flag;
17 }
```

二分的三种情况

```
1  //1 满足等于x的位置
2  int check(int a[],int x){
3
4      int l=1,r=n;
5      while(l<=r){
6          int mid=(l+r)>>1;
7          if(a[mid]<=x)
8              l=mid+1,ans=mid;
9          else//>= 右侧可能是答案
10             r=mid-1;
11     }
12     return ans;
13 }
```

二分的三种情况

```
1  //2 满足大于等于x 最靠左的位置
2  int check(int a[],int x){
3      if(a[n]<x)return n+1;
4      int l=1,r=n;
5      while(l<r){
6          int mid=(l+r)>>1;//取中心靠左位置
7          if(a[mid]<x)
8              l=mid+1;
9          else//>= 右侧可能是答案
10             r=mid;
11     }
12     return l;
13 }
```

二分的三种情况

```
1  //3 满足小于等于x 最靠右的位置
2  int check(int a[],int x){
3      if(a[n]<x)return n+1;
4      int l=1,r=n;
5      while(l<r){
6          int mid=(l+r+1)>>1;//取中心靠近右端点
7          if(a[mid]<=x)
8              l=mid;//左侧可能是答案
9          else//右侧不可能是答案
10             r=mid-1;
11     }
12     return l;
13 }
```

```
4  int check(int le, int ri,int x){
5      int mid,ans;
6      if(x>a[n])
7          return n+1;
8      while(le<=ri){
9          mid=(le+ri)/2;
10         if(a[mid]>=x){
11             ans=mid;
12             ri=mid-1;
13         }
14         else le=mid+1;
15     }
16     return ans;
17 }
```

```
19 int check(int le, int ri,int x){
20     int mid,ans;
21     while(le<ri){
22         mid=(le+ri)/2;
23         if(x>=a[mid]){
24             ri=mid;
25         }
26         else le=mid+1;
27     }
28     return le;
29 }
```

1.3 [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，那么，这个最大的最小距离是什么呢

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	样例输出
5 3 1 2 8 4 9	3

题解：

（1）暴力法。从小到大枚举最小距离的值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：第一头牛放在x1，第二头牛放在xj≥x1+dis的点xi,第三头牛放在xk≥xj+dis的点xk，等等，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。分析从小到大检查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)很快
                                    //优化：把二分上限设置为1e9
    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;            //牛棚不够，缩小距离
    }
    cout << ans;                    //打印答案
    return 0;
}
```

[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
```

题解：

容易发现，对于给定的序列，n越大能过的题是越少的，所以可以二分来求刚好过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。

```cpp
#include <bits/stdc++.h>
using namespace std;

typedef long long LL;
const int N = 100010;
int n, k;
int a[N];
LL l, r;
LL ans1, ans2;

int check(LL u) {
    LL cnt = 0, s = 0;
    for (int i = 1; i <= n; i ++) {
        s += a[i];//当前的累计值n
        if (s < 0)
            s = 0;
        else if (s >= u){
            s = 0;//重新计数
            cnt ++ ;
        }
    }
    return cnt;
}
```

[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

$0 \le k \le 106, |s| \le 106$

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！

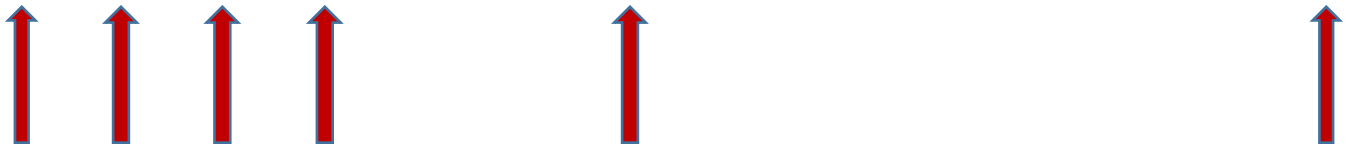
思路分析-枚举优化

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
```

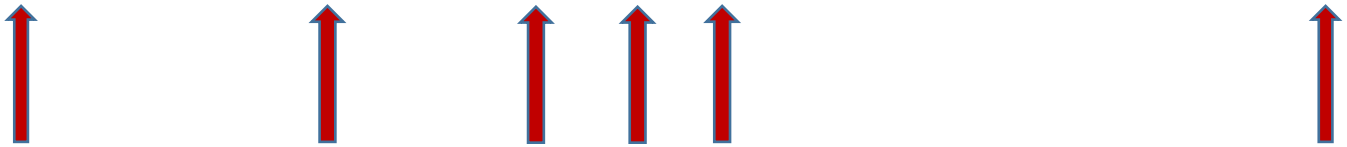
要下取整

```
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

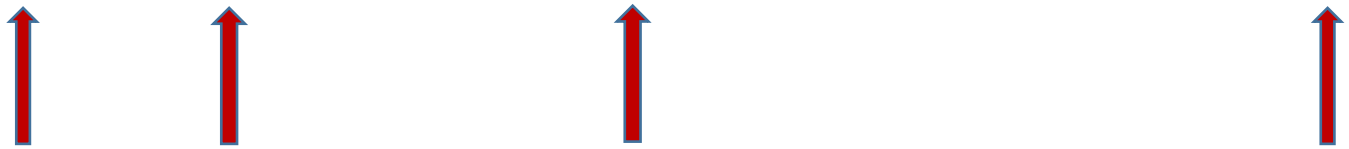
要上取整

```
    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;
}
```

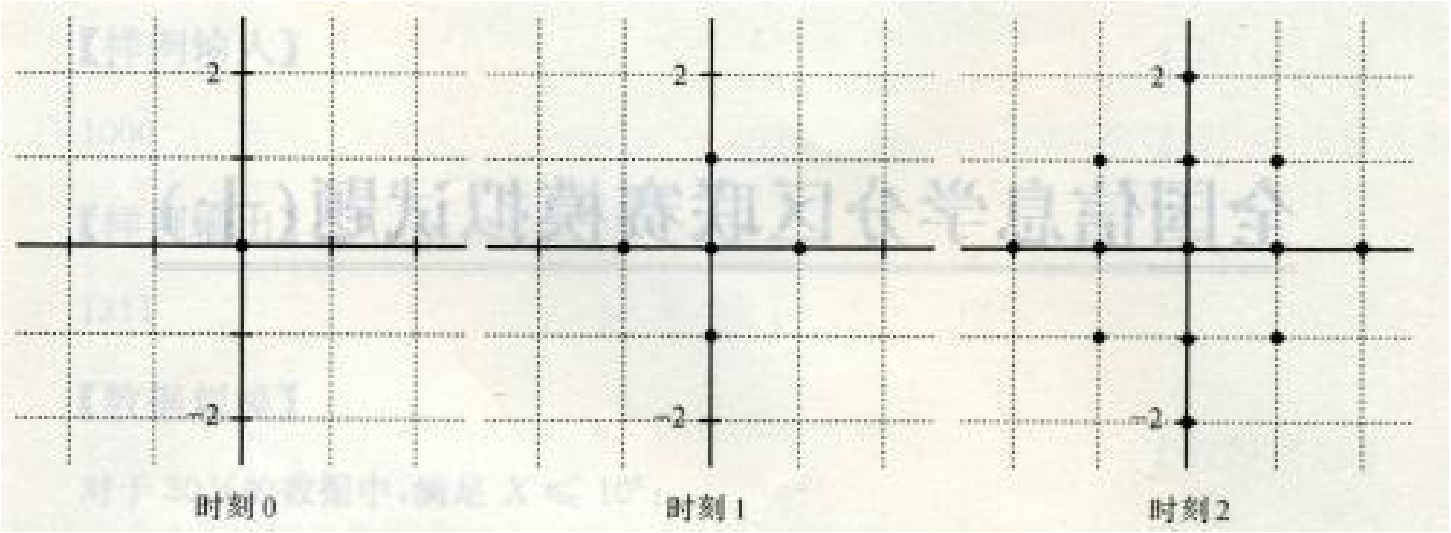
```
const int N = 1e6 + 10;
char s[N];//输入的原始数据
int sum[N];//存储1的前缀和
int k;//子串中包含1的个数
int main() {
    scanf("%d", &k);
    cin >> s + 1;
    int n = strlen(s + 1);
    if (k == 0) {
        int pos = 0, ans = 0;
        //特判k=0
        for (int i = 1; i <= n; i++) {
            if (s[i] == '0') {
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

1.3 [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;



```
    for (int i = 1; i <= n; i ++)
        scanf("%d", &a[i]);
    ans1 = -1e18, ans2 = 1e18;

    l = 1, r = 1e18;

    while (l <= r) {//求最大值
        LL mid = l + r >> 1;
        if (check(mid) >= k)
            ans1 = mid, l = mid + 1;
        else
            r = mid - 1;
    }

    l = 1, r = 1e18;

    while (l <= r) {//求最小值
        LL mid = l + r >> 1;

        if (check(mid) <= k)
            ans2 = mid, r = mid - 1;
        else
            l = mid + 1;
    }
    if (ans2 <= ans1)
        printf("%lld %lld\n", ans2, ans1);
    else
        puts("-1");

    return 0;
}
```

并查集+二分

在确定的范围为二分(mid)，然后检测在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。

```cpp
#include<iostream>
#include<cstdio>
#include<cmath>
using namespace std;
int xs[51],ys[51];//坐标
int father[51];//并查集 + 二分

int find(int i){//找根且合并
    if(father[i]==i)return i;
    return (father[i]=find(father[i]));
}

int main(){
    int n;
    cin>>n;
    for(int i=0;i<n;i++)cin>>xs[i]>>ys[i];
    int l=0,r=1000000000;
    int ans=0;//最终答案
    while(l<=r){
    cout<<ans<<endl;
    return(0);
}
```

```
while(l<=r){
    int mid=(l+r)>>1;//二分答案
    for(int i=0;i<n;i++){
        father[i]=i;
    }//初始化并查集

    for( int i=0;i<n;i++){
        for( int j=i+1;j<n;j++){
            int dis=abs(xs[i]-xs[j])+abs(ys[i]-ys[j]);
            if(dis<=mid*2){//能扩散到就连边
                //两个点都能扩散，所以相对的扩散速度会增倍
                int xx=find(i),yy=find(j);
                if(xx!=yy)father[xx]=yy;//合并
            }
        }
    }
```

```
35          int cnt=0;//连通块个数
36          for( int i=0;i<n;i++){
37              if(father[i]==i)cnt++;
38          }//几个根就是几个连通块
39          if(cnt==1){
40              ans=mid;
41              r=mid-1;
42          }//只有一个连通块就更新答案向下查找
43          else{
44              l=mid+1;
45          }
46      }
47      cout<<ans<<endl;
48      return 0;
49  }
```

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

```
1  #include<stdio.h>
2  #include<math.h>
3  double PI = acos(-1.0);//3.141592653589793;
4  #define eps 1e-5
5  double area[10010];
6  int n,m;
7  bool check(double mid){
8      int sum = 0;
9      for(int i=0;i<n;i++) //把每个披萨都按大小mid分开。统计总数
10         sum += (int)(area[i] / mid);
11     if(sum >= m) return true;   //最后看总数够不够m个
12     else         return false;
13 }
```

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！