第 12 章 二分

1 二分查找、二分查找边界

一、二分查找

1. 二分查找简介

二分查找也称折半查找（Binary Search），它是一种效率较高的查找方法。但是，折半查找要求线性表必须采用顺序存储结构，而且表中**元素按关键字有序排列**。

2. 二分查找时间复杂度：$\log_2 n$。

推导：

因为二分查找每次排除掉一半的不适合值，所以对于 n 个元素的情况：

一次二分剩下：n/2

两次二分剩下：n/2/2 = n/4

…

m 次二分剩下：n/(2^m)

在最坏情况下是在排除到只剩下最后一个值之后得到结果，即

n/(2^m)=1

所以由上式可得 ： 2^m=n

进而可求出时间复杂度为： $\log_2(n)$

注意：

Log2(1,000,000) ≈ 19.9

Log2(100,000,000) ≈ 26.6

3. 二分查找常见问题

（1）二分查找元素 x 在数列中是否存在（求出现位置）

（2）二分查找左边界：第一次出现的位置（>=该元素的第 1 个数的位置）

（3）二分查找左边界：第一次出现的位置（>=该元素的第 1 个数的位置）

1236：【基础】二分查找

	10	30	45	50	100	120	121	300	500	600
0	1	2	3	4	5	6	7	8	9	10

假设分别查找：50、5、600、150、800

解法一：闭区间[l,r]

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

//常量
const int N = 1000100;//数据量
int a[N];
int x,n;

int main(){
    //cin>>n;
    scanf("%d",&n);
    for(int i = 1;i <= n;i++){
        scanf("%d",&a[i]);
    }
```

```
    scanf("%d",&x);

    //二分
    //交代二分的边界
    int l = 1,r = n,mid;
    //如果区间内有值[l,r]
    while(l <= r){
        mid = (l + r) / 2;//中间元素的下标
        //分情况讨论
        if(x < a[mid]) r = mid - 1;
        else if(x > a[mid]) l = mid + 1;
        else if(x == a[mid]){
            //cout<<mid;
            printf("%d",mid);
            return 0;
        }
    }

    //找不到
    //cout<<-1;
    printf("%d",-1);
    return 0;
}
```

注意：求 mid 的 3 种写法

（1）mid = (l + r) / 2

（2）mid = (l+r) >> 1（括号可以不写）

（3）mid = l + (r - l) / 2 （可以防止 l+r 越界的情况）

解法二：左闭右开区间[l,r)

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

//常量
const int N = 1000100;//数据量
int a[N];
int x,n;

int main(){
    //cin>>n;
    scanf("%d",&n);
    for(int i = 1;i <= n;i++){
        scanf("%d",&a[i]);
    }
    scanf("%d",&x);

    //二分
    //交代二分的边界
    //左闭右开的写法，r 不在查找范围内
    int l = 1,r = n + 1,mid;
    //如果区间内有值[l,r)
    while(l < r){
        mid = (l + r) >> 1;//中间元素的下标
        //分情况讨论
        if(x < a[mid]) r = mid;//注意：r 不在讨论范围
        else if(x > a[mid]) l = mid + 1;
        else if(x == a[mid]){
            //cout<<mid;
            printf("%d",mid);
            return 0;
        }
    }
```

```
    //找不到
    //cout<<-1;
    printf("%d",-1);
    return 0;
}
```

注意：

左闭右开区间的好处：

1. 上下界之差等于元素的数量
2. 易于表示两个相邻子序列，一个子序列的上界就是另一个子序列的下界
3. 表达空集时，不会使得上界小于下界

二、二分查找边界

1894：【基础】二分查找左侧边界

	10	10	30	30	30	30	30	30	40	40
0	1	2	3	4	5	6	7	8	9	10

假设分别查找：30、10、40、15、50

二分查找左边界注意点：

（1）二分查找，如果 a[mid] == x，还要向左侧看，判断左侧是否还是 x；

（2）找左边界的本质：找数组中第一个>=x 的元素的位置；

（3）找到位置 L 之后，要判断 a[L]==x（如意，如果都是负数，找 0，要判断 L 在下标范围内）

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

const int N = 100100;
int a[N];
int n,q;//n 个数，q 次查询

//求元素 x 在数组 a 中首次出现的位置
int fun(int x){
    //确定边界
    int l = 1,r = n,mid;
    while(l <= r){
        mid = l + r >> 1;
        if(x < a[mid]) r = mid - 1;
        else if(x > a[mid]) l = mid + 1;
        else if(x == a[mid]) r = mid - 1;
    }

    if(a[l] == x) return l;
    else return -1;
}

int main(){
    cin>>n;
    for(int i = 1;i <= n;i++){
        cin>>a[i];
    }
    cin>>q;
    int x;
    //q 次查询
    for(int i = 1;i <= q;i++){
```

```
        cin>>x;
        //求元素 x 在数组 a 中首次出现的位置
        cout<<fun(x)<<" ";
    }
    return 0;
}
```

1895：【基础】二分查找右侧边界

	10	10	30	30	30	30	30	30	40	40
0	1	2	3	4	5	6	7	8	9	10

假设分别查找：30、10、40、15、50

二分查找右边界注意点：

（1）二分查找，如果 a[mid] == x，还要向右侧看，判断右侧是否还是 x；

（2）找右边界的本质：找的值 L，是数组中第一个 > x 的元素的位置；

（3）因此要判断 **L-1（或者 R）**的值是否和 x 相等（a[L-1]==x）；

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

const int N = 100100;
int a[N];
int n,q;

//找右边界
int fun(int x){
    int l = 1,r = n,mid;//查找的范围
    while(l <= r){
        mid = (l + r) >> 1;
        //分情况讨论
        if(x < a[mid]) r = mid - 1;
        else if(x > a[mid]) l = mid + 1;
        else if(x == a[mid]) l = mid + 1;//相等向右看
    }

    //l 就是数组中第一个>元素 x 的位置
    if(a[l-1] == x) return l-1;
    else return -1;
}

int main(){
    cin>>n;
    for(int i = 1;i <= n;i++){
        cin>>a[i];
    }
    cin>>q;
    //q 次询问
    int x;
    for(int i = 1;i <= q;i++){
        cin>>x;
        cout<<fun(x)<<" ";
    }
    return 0;
}
```

三、二分查找课堂练习

1898：【基础】同时出现的数

解法一：二分求解

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

/*
第 2 组数中的哪些数，在第 1 组数中出现了
从小到大输出所有满足条件的数

思路：采用二分查找第 2 个数组中哪些数在第 1 个数组中出现了
1.对 a 数组排序：方便二分查找
2.对 b 数组排序：方便结果从小到大输出
3.循环 b 数组的每个数，在 a 数组中二分查找是否存在
*/
const int N = 100010;
int a[N],b[N];
int n,m;

//在 a 数组中判断元素 x 是否存在
bool fun(int x){
    int l = 1,r = n,mid;
    while(l <= r){
        mid = l + r >> 1;
        if(x < a[mid]) r = mid - 1;
        else if(x > a[mid]) l = mid + 1;
        else return true;
    }

    return false;
}

int main(){
    cin>>n>>m;
    for(int i = 1;i <= n;i++){
        cin>>a[i];
    }
    for(int i = 1;i <= m;i++){
        cin>>b[i];
    }

    sort(a+1,a+n+1);
    sort(b+1,b+m+1);

    //循环 b 数组的每个数，逐个判断在 a 数组中是否存在
    for(int i = 1;i <= m;i++){
        if(fun(b[i])){
            cout<<b[i]<<" ";
        }
    }
    return 0;
}
```

解法二：map 求解

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

/*
第 2 组数中的哪些数，在第 1 组数中出现了
从小到大输出所有满足条件的数
*/
const int N = 100010;
int a[N],b[N];
int n,m;
map<int,int> ma;
```

```
int main(){
    cin>>n>>m;
    for(int i = 1;i <= n;i++){
        cin>>a[i];
        ma[a[i]] = 1;//标记出现过的数
    }
    for(int i = 1;i <= m;i++){
        cin>>b[i];
    }

    sort(b+1,b+m+1);

    //循环 b 数组的每个数，逐个判断在 a 数组中是否存在
    for(int i = 1;i <= m;i++){
        if(ma[b[i]] == 1){
            cout<<b[i]<<" ";
        }
    }
    return 0;
}
```

1899：【基础】最满意的方案

学校分数：

	100	150	310	350	380	390	400	420	450	480
0	1	2	3	4	5	6	7	8	9	10

假设某几位同学考试成绩为：150 分、320 分、90 分、500 分，应如何求解最满意的方案？

求左边界？右边界？

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

const int N = 100100;
int sc[N],x;
int m,n,s = 0;//s 最小差值的和

int main() {
    cin>>m>>n;
    //m 个学校
    for(int i = 1;i <= m;i++){
        cin>>sc[i];
    }
    //对学校分数排序，方便二分
    sort(sc+1,sc+m+1);

    //n 个同学
    int l,r,mid;
    for(int i = 1;i <= n;i++){
        cin>>x;//x：同学的分数
        //特判不需要二分的情况
        if(x <= sc[1]) s = s + sc[1] - x;
        else if(x >= sc[m]) s = s + x - sc[m];
        else{
            //找左边界
            l = 1;
            r = m;
            while(l <= r){
                mid = l + r >> 1;
                if(x <= sc[mid]) r = mid - 1;
```

```
                else l = mid + 1;
            }

            //看 l 和 l-1 哪个位置的分数差更小
            s = s + min(sc[l] - x,x - sc[l-1]);
        }
    }

    cout<<s;
    return 0;
}
```

注意：本题找有边界，将 if(x <= sc[mid]) 改成：if(x < sc[mid])即可。

四、二分查找作业

1896：【基础】二分查找满足条件的数

2078：【入门】起止位置

1542：【提高】小 X 算排名

1902：【提高】最少的修改次数

解决本题，需先掌握 LIS 问题（1893：【提高】最长上升子序列 LIS（2））。

2 二分函数

一、binary_search()：二分查找函数

1、binary_search(a+begin,a+end,x,cmp)；

函数含义：在 a 数组的下标为[begin,end)区间内，按照 cmp 的排序规则，找元素 x；找到返回 true，找不到返回 false。

注意点：

（1）查找区间是左闭右开的：[begin,end),不包含结束位置；

（2）排序规则 cmp 不是必须的，但查找时的排序规则，必须和排序的规则是一致的；

（3）等于的含义：a 等于 b 等价于 a 在 b 的前面，b 在 a 的前面都不成立

二、lower_bound()和 upper_bound()

1、lower_bound()：二分查找左边界

T* lower_bound(a+begin,a+end,x,cmp)；

函数含义：在 a 数组的下标为[begin,end)区间内，按照 cmp 的排序规则，找元素 x 的左边界（第一个 >= 元素的 x 的位置），返回位置指针；

注意点：

（1）基本注意点同 binary_search；

（2）此处返回的不是下标的值，而是返回指针：T* p；

（3）如果找不到符合条件的元素位置，指向下标为 end 的元素位置；

2、T* upper_bound：二分查找右边界

T* upper_bound(a+begin,a+end,x,cmp)；

函数含义：在 a 数组的下标为[begin,end)区间内，按照 cmp 的排序规则，找元素 x 的右边界（第一个 > 元素的 x 的位置），返回位置指针；

注意点：同 lower_bound；

例子：

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

//按个位数升序排序
bool cmp(int a1,int a2) {
    return a1%10 < a2%10;
}

void print(int a[],int n) {
    for(int i = 0; i < n; i++) {
        cout<<a[i]<<" ";
    }
    cout<<endl;
}

int main() {
    int n = 7;
    int a[7] = {12,5,3,5,98,21,7};
    sort(a,a+n);
    print(a,n);
    int *p = lower_bound(a,a+n,5);
    cout<<*p<<","<<p-a<<endl;//5,1
```

```
    p = upper_bound(a,a+n,5);
    cout<<*p<<endl;//7
    cout<<*upper_bound(a,a+n,13)<<endl;//21

    sort(a,a+n,cmp);
    print(a,n);  //21 12 3 5 5 7 98
    cout<<*lower_bound(a,a+n,16,cmp)<<endl;//7
    cout<<lower_bound(a,a+n,16,cmp)-a<<endl;//5(下标)

    cout<<upper_bound(a,a+n,18,cmp)-a<<endl;//7(下标)
    if(upper_bound(a,a+n,18,cmp)==a+n){
        cout<<"not found"<<endl;
    }

    cout<<*upper_bound(a,a+n,5,cmp)<<endl;//7
    cout<<*upper_bound(a,a+n,4,cmp)<<endl;//5

    return 0;
}
```

三、二分函数练习

1236：【基础】二分查找

1894：【基础】二分查找左侧边界

1895：【基础】二分查找右侧边界

1898：【基础】同时出现的数

1542：【提高】小 X 算排名

1542：【提高】小 X 算排名

1899：【基础】最满意的方案

3 二分答案

一、二分答案程序模板

二分答案与二分查找类似，也就是对有着单调性的答案进行二分，用于求解满足某种条件下的最大（小）值。

二分答案的常见模板：

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

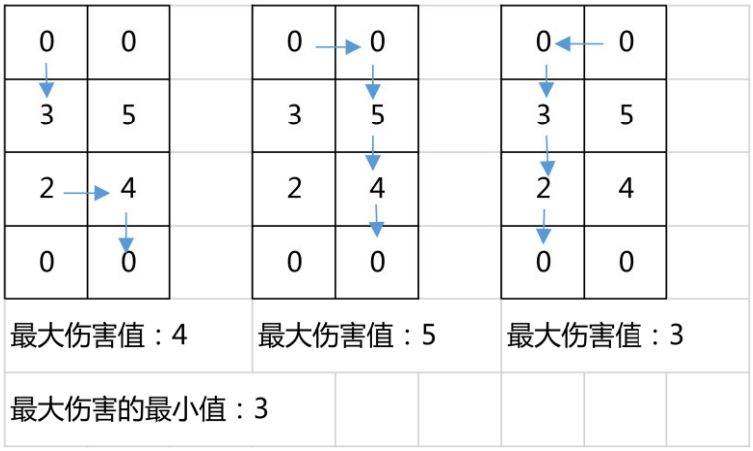
二、二分答案课堂练习

1908：【基础】伐木工

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

/*
本题：读入的数据在 int 范围，但求和可能会超过 int
因此定义 long long。
思考：找左边界，还是右边界；
本题：在高度为 mid 的情况下，如果能得到>=m 米的木材，升高高度
因此本题求右边界；
*/

const int N = 1000010;
long long a[N];
long long n,m,l = 1,r,mid;

//检验锯片高度为 x 的情况下，能够得到的木材量是否>=m
bool check(long long x){
    long long s = 0;//能够锯到的木材量
    for(int i = 1;i <= n;i++){
        //如果 x<树的高度，才能锯到木材
        if(x < a[i]) s = s + a[i] - x;
        //如果得到>=m 米的木材，就可以停止返回
        if(s >= m) return true;
    }

    return false;
}

int main(){
    cin>>n>>m;
    //读入木材的高度
    for(int i = 1;i <= n;i++){
        cin>>a[i];
        r = max(a[i],r);//r 的值应该是：所有树的最高高度
    }

    //二分答案
    while(l <= r){
        mid = l + r >> 1;
        //如果高度为 mid 的情况下，能够得到>=m 的木材
        if(check(mid)) l = mid + 1;
        else r = mid - 1;
    }

    cout<<l - 1;//右边界
```

```
    return 0;
}
```

1916: 【提高】防御迷阵




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

/*
第 1 行是入口，第 n 行是出口
伤害值：经过所有点的伤害值的最大
求：最小的伤害代价（求最大值的小的问题）

二分答案：
1.答案具备单调性——可以二分
2.思考二分的范围：答案在什么范围内
3.思考左边界、右边界
*/

int a[1010][1010];
int n,m;
int l = INT_MAX,r = INT_MIN,mid;
//广搜变量准备
int q[1000100][3];
int fx[5] = {0,0,1,0,-1};
int fy[5] = {0,1,0,-1,0};
bool f[1010][1010];//标记是否走过

//check(mid)：判断伤害值为 mid 的情况下，能否通过迷阵
//检验方法：采用广搜，从 1,1 出发，走 a[i][j]<=mid 的点，看能否走到第 n 行
bool check(int v){
    //重设标记数组的值，因为要广搜多次
    memset(f,false,sizeof(f));
    int h = 1,t = 1;//指针
    //交代起始点
    q[1][1] = 1;
    q[1][2] = 1;
    f[1][1] = true;//标记走过
    int tx,ty;//要去的点
    while(h <= t){
        //尝试四方向
        for(int i = 1;i <= 4;i++){
            tx = q[h][1] + fx[i];
            ty = q[h][2] + fy[i];

            //如果该点可行
            //!表示取反
            if(tx>=1&&tx<=n&&ty>=1&&ty<=m&&!f[tx][ty]&&a[tx][ty]<=v){
                t++;
                q[t][1] = tx;
                q[t][2] = ty;
```

```
                f[tx][ty] = true;//走过标记

                //判断是否出迷阵
                if(tx == n) return true;
            }
        }
        h++;
    }

    return false;
}

int main(){
    cin>>n>>m;
    for(int i = 1;i <= n;i++){
        for(int j = 1;j <= m;j++){
            cin>>a[i][j];
            //如果不是第 1 行和最后一行，才是伤害值
            if(i != 1 && i != n){
                l = min(l,a[i][j]);
                r = max(r,a[i][j]);
            }
        }
    }

    //二分
    while(l <= r){
        mid = l + r >> 1;
        //如果伤害值为 mid，能通过，减少伤害值，再尝试
        if(check(mid)) r = mid - 1;
        else l = mid + 1;
    }

    cout<<l;//左边界
    return 0;
}
```

1909：【提高】跳石头

直接考虑问题比较困难。我们首先考虑另一个问题，如果给定一个距离 mid， 问至少要移走多少石头才能满足石头之间的最小距离不小于 mid？对于这个问题， 答案就很简单了。我们采取贪心的策略计算：从左岸开始，移走它小于 d 的所有石头，再往后跳一步，循环往复。所以我么将石头按位置排序，在模拟一遍跳的过程就可以得到这个问题的答案。

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

/*
在起点和终点之间，有 N 块岩石
最短跳跃距离越大，搬走的石块就越多！
考虑极端情况，如果跳跃距离为 L，都要搬走！
如果跳跃距离为 1，都不用搬！
1.二分对象：跳跃距离，在[1,l]之间
2.最短跳跃距离为 mid 的情况下，如何检验？
检验搬走石头的数量：c！
如果 c<=m，说明最短跳跃距离太小，要放大
否则，要缩小！
3.求的是右边界
*/

const int N = 50100;
int a[N];
int n,m,l;//n 个石头，搬走 m 个，河道长 l

//求：最短跳跃距离为 mid 的情况下，搬走的石块数量是否<=m
```

```
bool check(int mid){
    int c = 0;//搬走的数量
    int p = 0;//人的位置
    //讨论所有的石块
    for(int i = 1;i <= n;i++){
        //跳跃距离比 mid 小，不符合要求，移走石块，人不动
        if(a[i] - p < mid) c++;
        else p = a[i];//人跳过去
    }

    //判断最后一次跳跃
    if(l - p < mid) c++;

    return c <= m;
}

int main(){
    cin>>l>>n>>m;
    //读入石头
    for(int i = 1;i <= n;i++){
        cin>>a[i];
    }

    //二分
    int left = 1,right = l,mid;
    while(left <= right){
        mid = left + right >> 1;
        //如果：最短跳跃距离为 mid 的情况下，搬走的石块数量<=m，放大距离
        if(check(mid)) left = mid + 1;
        else right = mid - 1;
    }

    cout<<left - 1;//右边界
    return 0;
}
```

三、二分答案练习

1910：【基础】愤怒的奶牛

1912：【基础】最小的空旷指数