算法引入

生活中处处会用到二分。

小时候，我们会玩一种猜数的游戏，一个人（以下简称玩家1）在心中默想一个数（以下简称x），另一个人（以下简称玩家2）则通过询问来尝试x。而玩家1每次只能对玩家2所猜测的数进行三个回答：

- 回答大于x
- 回答小于x
- 回答等于x

我们暂且把范围选定为$[1,1000]$。那么，当我们盲目地猜测的时候，最多的猜测次数（即最坏情况的次数）为1000次。

很明显，这样的猜测是毫无意义的。于是，必须要改变猜测的方法。

我们其实可以在一开始从区间的一半的位置开始，如果回答（即在这个例子中的500）大于x，我们就可以排除大于500的部分，只看$[1,500)$范围内的整数。反之，如果小于500，就看$(500,1000]$的部分。接着我们如法炮制，不断地缩小数的范围，每次都询问这个范围内最中间的数（是小数的话可以取整）。这样，每一次的范围都可以缩小到原来的$\frac{1}{2}$。

也就是说，对于任意一个区间，我们只需要不断地将区间缩为原来的$\frac{1}{2}$，直到范围缩减为1，即区间$[x,x]$。这样，我们就能够完成对数的猜测。

当然，有的时候我们的运气会较好一些，可能在范围还没到1的时候就正好猜到了x。但是为了保险，我们可以来计算一下最坏情况的猜测次数。

对于一个区间$[l,r]$，设其整数元素个数有m个。它所表示的集合总共有$r-l+1$个数。每一次猜测，都可以将区间缩小为原来的$\frac{1}{2}$。

对于任意一个实数r，$log_n(r)$是满足$n^{log_n(r)}=r$的实数。因此，由于每一次区间的元素个数会除以2，所以经过$\lceil log_2(m)\rceil$次后，区间的大小会变为$\frac{m}{2^{\lceil log_2(m)\rceil}}$。

然而，经过上述操作之后，如果每一次都直接除以2（即不整除），则区间的大小必定≤ 1。因为实际上进行的是整除，所以最终可能猜测次数要大于这个数。但在任何情况下，总的猜测次数必定**大于等于**这个数，即猜测次数最多为：

$$\lceil log_2(r-l+1)\rceil$$

回到上面的例子，如果区间为$[1,1000]$，则$log_2(1000) \approx 9.96578$，所以猜测次数最多为$10$次。这远远小于$1000$！

生活中，铁轨的检测等都可能会用到二分。

代码实现

体验了生活中的二分之后，我们来写几个程序。

No.1 猜数程序

【要求】输入区间$[l,r]$（l,r都在$(-2^{63},2^{63}]$范围之内）。每次程序会进行询问。

如果询问大于被猜测数，用户就输入HIGH。如果询问小于被猜测数，就输入LOW。如果询问正确，则输入YES。输入错误就输出Error!。

最后输出猜测次数。

时间复杂度：$\mathcal{O}(log_{r-l+1})$

【实现】

```
#include<bits/stdc++.h>
using namespace std;
long long l,r,mid;//需要开long long
int cnt;
string result;
int main()
{
    scanf("%lld%lld",&l,&r);
    if(l>r)swap(l,r);//默认小的为l，大的为r
    while(l<=r)//只有在左指针小于等于右指针的时候才能继续执行
    {
        mid=(l+r)>>1;//每一次mid设置为中间数
        printf("%lld\n",mid);//询问
        cnt++;//猜测次数加1
        while(cin>>result)
        {
            if(result=="YES")//正确就输出猜测次数并结束程序
            {
                printf("%d\n",cnt);
                return 0;
            }
            if(result=="HIGH")//询问高于原数就把右指针往左移
            {
                r=mid-1;
                break;
            }
            else if(result=="LOW")//询问低于原数就把左端点往右移
            {
                l=mid+1;
                break;
            }
            else puts("Error!");//报错
        }
    }
    return 0;
}
```

当然，上述代码是较为简单的，如果玩家故意改变被猜测数，那么程序就会一直执行下去。

No.2 查找

题目链接

本题是二分查找的一道模板题。

本题的要求是输入数组然后在数组中查找对应的数。相比猜数游戏来说，我们缩小的是数组区间的范围。之前是从$[1,1000]$这1000个数进行二分，而现在我们可以从$[1,n]$的范围开始，二分$a_1,...,a_n$这

n个元素。查找到输出位置即可。但是，这必须要在数组从小到大排列的情况下进行（即本题中的单调不减）。

【方法1】暴力法

按照题目模拟查找即可。对于每一个询问，进行查找即可。如果大于元素就直接返回-1。

时间复杂度：$\mathcal{O}(nm)$，最大可达到10^{11}

期望得分：0分（全部TLE）

代码如下：

```
#include<bits/stdc++.h>
using namespace std;
int n,m,a[1000001];
int find(int x)
{
    for(int i=1;i<=n;i++)//暴力搜索
    {
        if(a[i]==x)return i;
        if(a[i]>x)return -1;
    }
    return -1;
}
int read()
{
    int x=0,f=1;
    char ch=getchar();
    while(ch<'0'||ch>'9')
    {
        if(ch=='-')f=-1;
        ch=getchar();
    }
    while(ch>='0'&&ch<='9')
    {
        x=(x<<1)+(x<<3)+(ch^48);
        ch=getchar();
    }
    return x*f;
}
int main()
{
    cin>>n>>m;
    for(int i=1;i<=n;i++)a[i]=read();
    while(m--)printf("%d ",find(read()));
    return 0;
}
```

【方法2】朴素二分实现（从右到左）

我们可以使用二分优化。每一次找到的都是最右边满足题意的数，所以每次需要往前推。效率略有提高。

时间复杂度：$\mathcal{O}(m(log_n+n))$（这在大部分情况下优于前面一种，但在极端的情况下比第一种算法还要慢）

期望得分：80分（一个测试点TLE）

```cpp
#include<bits/stdc++.h>
using namespace std;
int read()
{
    int x=0,f=1;
    char ch=getchar();
    while(ch<'0'||ch>'9')
    {
        if(ch=='-')f=-1;
        ch=getchar();
    }
    while(ch>='0'&&ch<='9')
    {
        x=(x<<1)+(x<<3)+(ch^48);
        ch=getchar();
    }
    return x*f;
}
int n,m,a[1000001];
int main()
{
    n=read();
    m=read();
    for(int i=1;i<=n;i++)a[i]=read();
    while(m--)
    {
        int q=read(),l=1,r=n,mid,ans=-1;
        while(l<=r)
        {
            mid=l+r>>1;
            if(a[mid]==q)
            {
                ans=mid;
                break;
            }
            else if(a[mid]<q)l=mid+1;
            else r=mid-1;
        }
        for(int i=ans-1;i;i--)
        {
            if(a[i]!=q)
            {
                ans=i+1;
                break;
            }
        }
        printf("%d ",ans);
    }
    return 0;
}
```

【方法3】朴素二分实现（从左到右）

时间复杂度：$\mathcal{O}(mlog_n)$，最大可以达到2×10^6

期望得分：100分

```
#include<bits/stdc++.h>
using namespace std;
int read()//输入的数较多，可用快读
{
    int x=0,f=1;
    char ch=getchar();
    while(ch<'0'||ch>'9')
    {
        if(ch=='-')f=-1;
        ch=getchar();
    }
    while(ch>='0'&&ch<='9')
    {
        x=(x<<1)+(x<<3)+(ch^48);
        ch=getchar();
    }
    return x*f;
}
int n,m,a[1000001];
int find(int x)//查找函数
{
    int l=1,r=n,mid;//区间一开始为[1,n]，mid为中间位置（即数组下标）
    while(l<r)
    {
        mid=l+((r-l)>>1);
        //如果上一个区间为[2,6]，则这一次将查找2+(6-2)÷2=4，即a[4]的值
        if(a[mid]>=x)r=mid;//如果该位置大于等于被询问的数，则右指针变为该位置，即在上述例子中，区间变为
        else l=mid+1;//否则左指针+1，即往右侧逼近
    }
    return a[l]==x?l:-1;//如果a[l]等于x，则说明已经找到了最左侧等于x的数，否则就返回-1，表示没有找到
}
int main()
{
    n=read();
    m=read();
    for(int i=1;i<=n;i++)a[i]=read();
    while(m--)printf("%d ",find(read()));//每一次询问，输出答案即可
    return 0;
}
```

【方法4】STL实现

在C++中，STL经常会被使用到，而且有简化代码的作用（~~虽然有的时候时间消耗会大一些~~）。这里就可以使用一个函数lower_bound()。

对于该该函数，我们可以查找一个数组中第一个大于等于某数的元素位置，即在$[first, last)$中。但是被操作的数组必须严格单调不减。

注意到上述代码中要减去a！原因是该函数返回的不是数组的下标，而是一个指针。所以最后减去a就是它所表示的下标。特别地，如果查找不到这个元素，则函数返回是$last$的值。

例如：

```
int a[]={1,2,3,5,7,8};
cout<<lower_bound(a,a+6,6)-a;
//第一个大于等于6的数是7，所以输出4
cout<<lower_bound(a,a+6,9)-a;
//第一个大于等于9的数不存在，所以输出6
```

对于本题来说，找到该数组元素后，如果它不等于被查找数，那么就输出-1，否则输出该下标。

时间复杂度：$\mathcal{O}(mlog_n)$，最大可达到2×10^6

期望得分：100分

```
#include<bits/stdc++.h>
using namespace std;
int n,m,a[1000001];
int read()
{
    int x=0,f=1;
    char ch=getchar();
    while(ch<'0'||ch>'9')
    {
        if(ch=='-')f=-1;
        ch=getchar();
    }
    while(ch>='0'&&ch<='9')
    {
        x=(x<<1)+(x<<3)+(ch^48);
        ch=getchar();
    }
    return x*f;
}
int main()
{
    n=read();
    m=read();
    for(int i=1;i<=n;i++)a[i]=read();
    while(m--)
    {
        int temp=read(),pos=lower_bound(a+1,a+n+1,temp)-a;//temp为被测数，而pos为满足条件的第一个元
        if(pos==n+1)//判断没有找到
        {
            printf("-1 ");
            continue;
        }
        if(a[pos]!=temp)//判断找到的不等于被查找数
        {
            printf("-1 ");
            continue;
        }
        printf("%d ",pos);//否则输出位置
    }
    return 0;
}
```

期望得分：100分

No.3 A-B数对

题目链接

【方法1】暴力实现

我们可以进行一个双重循环，如果找到满足的数对，那么计数器就加1。

时间复杂度：$\mathcal{O}(n^2)$，最大可达到4×10^{10}

期望得分：76分（三个测试点TLE）

```
#include<cstdio>
long long n,c,s,a[200001];
int main()
{
    scanf("%lld%lld",&n,&c);
    for(int i=1;i<=n;i++)scanf("%lld",&a[i]);
    for(int i=1;i<n;i++)
    {
        long long k1=c+a[i],k2=a[i]-c;
        for(int j=i+1;j<=n;j++)if(a[j]==k1||a[j]==k2)s++;
    }
    printf("%lld",s);
}
```

【方法2】朴素二分实现（算法精髓）

我们只需要找到一个等于（即大于等于）被查找数的元素下标，以及大于被操作数元素下标即可。计数器累加的应该是两下标之差。

下面是寻找大于等于被查找数的函数：

```
int find1(int x,int l,int r)
{
    int mid;
    while(l<=r)
    {
        mid=(l+r)>>1;
        if(a[mid]>=x)r=mid-1;
        else l=mid+1;
    }
    return l;
}
```

查找大于的函数：

```cpp
int find2(int x,int l,int r)
{
    int mid;
    while(l<=r)
    {
        mid=(l+r)>>1;
        if(a[mid]>x)r=mid-1;
        else l=mid+1;
    }
    return l;
}
```

我们将所有数据输入后，再跑一遍$[1,n]$的循环，然后把当前元素后面的所有数（不包括当前元素）进行二分，然后计数器累加即可。

时间复杂度：$\mathcal{O}(nlog_n)$，最大可达到约3.6×10^6

期望得分：100分

```
#include<bits/stdc++.h>
using namespace std;
int n,c;
long long ans,a[200001];
int find1(int x,int l,int r)
{
    int mid;
    while(l<=r)
    {
        mid=(l+r)>>1;
        if(a[mid]>=x)r=mid-1;
        else l=mid+1;
    }
    return l;
}
int find2(int x,int l,int r)
{
    int mid;
    while(l<=r)
    {
        mid=(l+r)>>1;
        if(a[mid]>x)r=mid-1;
        else l=mid+1;
    }
    return l;
}
int read()
{
    int x=0,f=1;
    char ch=getchar();
    while(ch<'0'||ch>'9')
    {
        if(ch=='-')f=-1;
        ch=getchar();
    }
    while(ch>='0'&&ch<='9')
    {
        x=(x<<1)+(x<<3)+(ch^48);
        ch=getchar();
    }
    return x*f;
}
int main()
{
    scanf("%d%d",&n,&c);
    for(int i=1;i<=n;i++)a[i]=read();
    sort(a+1,a+n+1);//快排来把数组单调不减，以实现二分
    for(int i=1;i<=n;i++)ans+=find2(a[i]+c,i+1,n)-find1(a[i]+c,i+1,n);
    //对于每一个i，区间为(i,n]，被查找数为a[i]+c，即A-B=C中，B为a[i]，即要寻找B+C
    printf("%lld",ans);
```

```
    return 0;
}
```

【方法3】STL:upper_bound和STL:lower_bound的结合实现

刚才我们介绍了lower_bound()，而upper_bound()于其唯一的区别就是查找的是大于被查找数的第一个元素。

令B为数组中的第一个元素，则只需要找到值为$B+C$元素的个数即可。

我们先快排一下数组，看一个例子：

```
int a[]={0,1,4,6,6,8,9};//为了方便，用0占位
int c=5;
```

对于$c=5$，我们先对a_1操作，即寻找$1+5=6$。第一个大于等于6的位置是3，而一个大于6的位置是5，所以满足的个数有$5-3=2$个。

对于a_6这个元素，要寻找$9+5=14$。因为找不到该元素，所以两个函数都返回的是7，而计数器会加$7-7=0$，所以找不到元素是不需要特判的。

时间复杂度：$\mathcal{O}(nlogn)$，最大可达到约3.6×10^6

期望得分：100分

```
#include<bits/stdc++.h>
using namespace std;
int n,c;
long long ans,a[200001];
long long read()
{
    long long x=0,f=1;
    char ch=getchar();
    while(ch<'0'||ch>'9')
    {
        if(ch=='-')f=-1;
        ch=getchar();
    }
    while(ch>='0'&&ch<='9')
    {
        x=(x<<1)+(x<<3)+(ch^48);
        ch=getchar();
    }
    return x*f;
}
int main()
{
    scanf("%d%d",&n,&c);
    for(int i=1;i<=n;i++)a[i]=read();
    sort(a+1,a+n+1);
    for(int i=1;i<=n;i++)ans+=upper_bound(a+1,a+n+1,a[i]+c)-lower_bound(a+1,a+n+1,a[i]+c);
    printf("%lld",ans);
    return 0;
}
```

【方法4】STL:map映射实现（投机取巧，不适合训练二分算法）

本方法在该题题解中也提到了。map简单来说，能够实现任意类型之间的一一对应，而且不必太考虑大小问题。

题目要求$A-B=C$数对的个数，那么我们就不妨改为$A-C=B$的个数。定义一个<long long,int>（因为不开long long会爆掉），即long long对应int的map，名称为m。一开始输入$a_1,...,a_n$，对于每一个数组元素，让$m[a_i]$的值加1。

然后跑一遍$[1,n]$循环，看m_{a_i-c}的值是多少。计数器加上该值即可。

时间复杂度：$\mathcal{O}(n)$，最大可达到约2×10^5

期望得分：100分

```
#include<bits/stdc++.h>
using namespace std;
int n,c;
long long ans,a[200001];
map<long long,int>m;//定义映射map
long long read()
{
    long long x=0,f=1;
    char ch=getchar();
    while(ch<'0'||ch>'9')
    {
        if(ch=='-')f=-1;
        ch=getchar();
    }
    while(ch>='0'&&ch<='9')
    {
        x=(x<<1)+(x<<3)+(ch^48);
        ch=getchar();
    }
    return x*f;
}
int main()
{
    scanf("%d%d",&n,&c);
    for(int i=1;i<=n;i++)
    {
        a[i]=read();
        m[a[i]]++;//对应下标为a[i]的元素加1
    }
    for(int i=1;i<=n;i++)ans+=m[a[i]-c];//计数器加上满足条件的个数
    printf("%lld",ans);
    return 0;
}
```

No.4 一元三次方程求解

题目链接

【方法1】暴力实现

时间复杂度：$\mathcal{O}(20000)$

期望得分：100分

```cpp
#include<bits/stdc++.h>
using namespace std;
double a,b,c,d,ans,x;
int main()
{
    cin>>a>>b>>c>>d;
    for(x=-100;x<=100;x+=0.01)
    {
        ans=a*x*x*x+b*x*x+c*x+d;
        if(fabs(ans)<1e-7)printf("%.2lf ",x);
    }
    return 0;
}
```

【方法2】二分实现

时间复杂度：$\mathcal{O}(200 \times log_{100})$

期望得分：100分

```
#include<bits/stdc++.h>
using namespace std;
double a,b,c,d;
double ans(double x)
{
    return a*x*x*x+b*x*x+c*x+d;
}
void search(double l,double r)
{
    if(r-l<=1e-3)
    {
        printf("%.2lf ",l);
        return;
    }
    double mid=l+(r-l)*0.5,ans1=ans(l)*ans(mid),ans2=ans(mid)*ans(r);
    if(!ans(mid))printf("%.2lf ",mid);
    if(!ans(r))printf("%.2lf ",r);
    if(ans1<0)search(l,mid);
    else if(ans2<0)search(mid,r);
}
int main()
{
    cin>>a>>b>>c>>d;
    for(int i=-100;i<100;i++)
    {
        if(ans(i)*ans(i+1)>0)continue;
        search(i,i+1);
    }
    return 0;
}
```

【方法3】盛金公式

盛金公式是一种求解一元三次方程的方法。

我们只需要考虑**盛金公式4**，即：

$$\begin{cases} A = b^2 - 3ac \\ B = bc - 9ad \\ C = c^2 - 3bd \end{cases}$$

$$X_1 = \frac{-b - 2\sqrt{A}cos\frac{\theta}{3}}{3a}$$

$$X_{2,3} = \frac{-b + \sqrt{A}(cos\frac{\theta}{3} \pm \sqrt{3}sin\frac{\theta}{3})}{3a}$$

其中$\theta=\arccos T$，且：

$$T=\frac{2Ab-3aB}{2\sqrt{A^3}}(A>0,-1<T<1)$$

期望得分：100分

```
#include<bits/stdc++.h>
using namespace std;
double a,b,c,d,A,B,C,T,theta,x[4];
int main()
{
    cin>>a>>b>>c>>d;
    A=b*b-3*a*c;
    B=b*c-9*a*d;
    C=c*c-3*b*d;
    T=(2*A*b-3*a*B)/(2*sqrt(A*A*A));
    theta=acos(T);
    x[1]=((-b-2*sqrt(A)*cos(theta/3)))/3/a;
    x[2]=(-b+sqrt(A)*(cos(theta/3)+sqrt(3)*sin(theta/3)))/3/a;
    x[3]=(-b+sqrt(A)*(cos(theta/3)-sqrt(3)*sin(theta/3)))/3/a;
    sort(x+1,x+4);
    printf("%.2lf %.2lf %.2lf",x[1],x[2],x[3]);
    return 0;
}
```

总结

总之，本次只介绍了二分最基本的用法。而二分这种思想，则将延续到后面的难题中，并得到充分的应用，因为二分是一种能优化查找效率的算法。

附加内容

以下题目/题单中的题目都部分/全部使用了二分算法，包括二分查找、二分排序、二分寻找答案等。

洛谷官方题单——【算法1-6】二分查找与二分答案

【模板】快速排序

[NOI Online #2 入门组]未了

本文章的所有无注释AC代码