尺取 二分 倍增

20数科胡露坤

部分课件参考 19信安张华睿师哥

查找一个数

A想一个1~1000中的数，让B来猜。B可以问问题，A只能回答是或否，最少需要多少次找到答案?

- 是否是1？是否是2？……
- 是否大于500？是否大于750？（是否大于250？）……

第二种方法每次缩小范围到上次的一半，至多10次就能找出答案。

二分

二分查找也称折半查找，其优点是查找速度快，缺点是要求所要查找的数据必须是**有序序列**。一般而言，对于包含n个元素的列表，用二分查找最多需要$logn$步，而简单查找最多需要n步。

> 该算法的基本思想是将所要查找的序列的中间位置的数据与所要查找的元素进行比较，如果相等，则表示查找成功，否则将以该位置为基准将所要查找的序列分为左右两部分。
> 接下来根据所要查找序列的升降序规律及中间元素与所查找元素的大小关系，来选择所要查找元素可能存在的那部分序列，对其采用同样的方法进行查找，直至能够确定所要查找的元素是否存在。

给出单调递增的整数序列$a[n]$，查找第一个大于或等于X的数的位置。

```
while(l<r){
        int mid=(l+r)>>1;    //l表示区间左端点，r表示区间右端点
        if(a[mid]>=x)     //x为目标值
                r=mid;
        else
                l=mid+1;
}
//区间[l,r)
```

l即为最终答案

另一种写法

```
while(l<r){
        int mid=(l+r+1)>>1;    //l表示区间左端点，r表示区间右端点
        if(a[mid]>=x)     //x为目标值
                r=mid-1;
        else
                l=mid;
}
```

lower_bound()

- 头文件#include<algorithm>
- 函数lower_bound(first,last,value)在[first,last)中进行二分查找，返回**大于或等于**value的第一个数的**地址**。所有数均小于value时返回last。

lower_bound() 基本用法

```
int a[]={1,3,5,7,9};
int val=6;
int res=lower_bound(a,a+5,val)-a;
//res==3 大于6的第一个数是7，下标3
```

- 注意函数最后返回的是地址，减去起始地址a才能得到最终下标。
- lower_bound的参数也可传入vector.begin()一类的迭代器。

upper_bound()

- 头文件#include<algorithm>
- lower_bound(first,last,value)，与lower_bound()类似，返回**大于**value的第一个数的**地址**。

lower_bound() 与 upper_bound()

```
int a[]={1,3,5,5,5,7,9};
int val=5;
int low=lower_bound(a,a+7,val)-a;
int up=upper_bound(a,a+7,val)-a;
```

输出low等于2，为第一个5的下标；

up等于5，为7的下标。

在实数域上的二分

求$f(x)=x^3-5x^2+10x-80=0$的根。($|f(a)|<=10^{-6}$即可)

- 简单分析可知f(x)单调递增，且在(0,100)中必有一根。
- 利用二分求解

代码

```
double eps = 1e-6;
while (fabs(y) > eps)
{
        if (y > 0)
                x2 = root;
        else
                x1 = root;
        root = x1 + (x2-x1)/2;//有时为了防止x1+x2溢出，可采取这种写法
        y = f(root);
}
```

适用情况

二分适用于答案具有一定单调性的情况。若正向求解比较复杂，而代入答案检验正确与否比较方便时，可以通过检验不断缩小区间，确定最终值。

尺取

顾名思义，像尺子一样取一段。尺取法通常是对数组保存一对下标，即所选取的区间的左右端点，然后根据实际情况不断地推进区间左右端点以得出答案。

例题

给定一个正数序列，求满足和大于或等于S的子序列（连续）的最短长度

分析

首先，序列都是正数，如果一个区间其和大于等于S了，那么不需要在向后推进右端点了，因为其和也肯定大于等于S但长度更长，所以，当区间和小于S时右端点向右移动，和大于等于S时，左端点向右移动以进一步找到最短的区间，如果右端点移动到区间末尾其和还不大于等于S，结束区间的枚举。

代码

```
while(1){
        while(r<n&&sum+a[r]<s){
                sum+=a[r++];
        }
        if(r==n)break;
        ans=min(ans,r-l+1);
        sum-=a[l++];
}
```

尺取法一般用于做具有单调性的，满足某一性质的区间问题。

倍增

倍增就是“成倍增长”。它能够使得线性的处理转化为对数级的处理，大大地优化时间复杂度，在很多算法中都有应用，如ST算法，LCA（最近公共祖先）等。
倍增利用了数的二进制表示。一个整数n，它的二进制表示只有$logn$位。如果要从0增长到n，可以用1、2、4、$\ldots$、2^k为“跳板”，快速增长到n。

例题

给定一个长度为n的数列A，然后进行若干次询问，每给定一个正整数T，求出最大的k,满足$\sum_{i=1}^{k} A[i] \leq T$。

```
cin>>n;
for(int i=1;i<=n;i++){
    cin>>A[i];
    S[i]=A[i]+S[i-1]; //S为A的前缀和
}
while(cin>>T){
    int p=1,k=0,sum=0;
    while(p!=0){
        if(S[k+p]<=T&&k+p<=n){
            k+=p;
            p*=2;//向后倍增
        }
        else
            p/=2;
    }
    cout<<k<<endl;
}
```

倍增RMQ（区间最值查询）

给定一个长度为n的数列，m次询问，每次询问给出区间[L,R]，求出每一次询问的区间内数字的最大值

- **ST（Sparse Table）算法**是一个非常有名的在线处理RMQ问题的算法，它可以在O(nlogn)时间内进行预处理，然后在O(1)时间内回答每个查询。

- 设$f[i][j]$表示从i开始,连续2^j个数中的最大值,即区间$[i, i+2^j-1]$中的最大值。
- 边界条件：$f[i,0]=A[i]$
- 递推公式:$f[i,j]=max(f[i,j-1],f[i+2^{j-1},j-1])$

```
//递推过程
for (int j = 1; j <= 20; ++j)
    for (int i = 1; i + (1 << j) - 1 <= n; ++i)
        f[i][j] = max(f[i][j - 1], f[i + (1 << j - 1)][j - 1]);
```

查询

当我们在询问时,肯定不能保证区间长度恰好是2的次幂,也就是和f[l][k]不一定重合。因此我们将区间$[l,r]$分成$[l,l+2^k],[r-2^k+1,r]$两个相交的部分。其中k是最大的满足2^k $\leq$区间长度的正整数。两部分再取max

即 $max(f[l][k],f[r-(1<<k)+1][k])$

AC愉快~