前言：博主在学习二分查找时，就对众多不同形式的二分查找写法感到疑惑，而且在OJ上,同一道题目，不同写法的二分，可能会让你 Wrong Answer 或者 TLE,,,,学校集训队有一个15学长，貌似队员都是用它的二分模板，而且基本没出过错，当时"太年轻”，总是觉得，为什么这学长写的二分模板这么健壮，为啥这么写才是对的呢， 当然当时也是没怎么想清楚，这几天，在图书馆，偶然翻到 <<编程珠玑>>, 发现里面对二分的探究，于是在网上学习了相关资料，但是感觉很多博文纯粹是为了写博文而写博文，写得很粗糙，于是趁机写篇关于二分博文，，希望可以帮助读者更好的理解二分查找。

第一个二分是在20世纪40年代出来的，但是第一个完全正确的二分直到20世纪60年代才出现，二分虽然简单，但是细节需要考虑很多，因此很多人写的二分都是有bug的

目录

首先二分查找的适用条件是什么？我们为什么要用二分查找?

There is no doubt that **当然是为了效率啊。**

我举个例子，在含有10^9个元素的非下降序列的数组（当然了，开这么大，会爆内存的）中查找某个特定值，并且我们要查找的元素刚好是第10^9个元素，且该值只存在一次。用从头到尾遍历的方法，我需要for循环10^9次才能查找到其下标。如果用

二分查找，每次循环将key值与区间的mid点的值比较，mid点的值刚好等于key值，就返回下标，否则更新左边界或者是右边界来使区间减半，这样每次待查区间就会不断减半，最终在某一次，区间会缩小到刚好mid点的值等于key值，或者区间的left端，或者区间的right端点就刚好指向key值（不同的二分写法 不一样，因此最后key值的下标如果存在的话，可能是通过mid，right，left端来指向的），这边二分的次数就是需要循环判断的次数，最多判断次数，也就是最坏复杂度, 也就是O(log2n);

备注：下面代码均是针对: 一个非递减数组，在其中查找key值，存在的话返回下标，不存在的话返回-1;

三个步骤写出正确的二分查找（先给出正确写法，，后面会分析错误代码）：

第一个步骤： 确定初始区间表示法。

我们都知道，一段有限区间的表示法，有4种：

左闭右闭: [0, 5]

左开右开 (-1, 6),

左闭右开 [0 , 6)

左开右闭 (-1, 5],

这四个区间都表示0 ,1 , 2 ,3 ,4，5这6个数的集合，同理，假如我们要表示一段下标为0.....5的数组，我们也可以用以上任意一种来表示。比较常用的是左闭右闭表示法，左闭右开表示法, 左开右开表示法。

如果选用的是左闭右开表示法来表示初始化区间:

```
int binary(int array[], int n, int v)
{
    int left, right, middle;

    left = 0, right = n;

    while ()
    {
        cout << "循环" << endl;
    }

    return -1;
}
```

如果选用的是左闭右闭表示法来表示初始化区间:

```
int binary(int array[], int n, int v)
{
    int left, right, middle;

    left = 0, right = n - 1;

    while ()
    {

        cout << "循环" << endl;
    }

    return -1;
}
```

如果选用的是左开右开表示法:

```
int binary(int array[], int n, int v)
{
    int left, right, middle;

    left = -1, right = n;

    while ()
    {

        cout << "循环" << endl;
```

```
    }

    return -1;
}
```

第二个步骤: 根据第一步所选的区间表示法，写出对应的循环条件：

如果选用的是左闭右开表示法来表示初始化区间:

```
int binary(int array[], int n, int v)
{
    int left, right, middle;

    left = 0, right = n ;

    while ( left < right )
    {
         cout << "循环" << endl;
    }
     return -1;
}
```

为什么对应的循环条件是这样呢? :其实很好理解的， **什么时候循环应该终止呢？当区间的实际长度已经<=0的时候。**

上面这样写，意味着循环终止的条件是left == right ,由于我们采用的是左闭右开表示法，也就是 区间为[left ,right) 且left == right 循环停止，，如果用个例子来解释，大家就会豁然开朗了， 假如[left=4，right=4），此时用左闭右闭来解释的话，区间实际上是出现了右边界=3 , 左边界=4，这种情形，右边界 < 左边界，这意味着区间长度<=0 了，无法继续二分了。

如果选用的是左闭右闭表示法来表示初始化区间:

```
int binary(int array[], int n, int v)
{
    int left, right, middle;

    left = 0, right = n - 1;

    while ( left <= right )
    {

        cout << "循环" << endl;
    }

    return -1;
}
```

上面这样写，意味着循环终止的条件是left > right ,由于我们采用的是左闭右闭表示法，也就是 区间为[left ,right],如果用个例子来解释，大家就会豁然开朗了， 假如[left=4， right=3]，区间实际上是出现了 右边界=3 ,左边界=4这种情形，也就是右边界 < 左边界，这意味着区间长度<=0 了，无法继续二分了。

如果选用的是左开右开表示法:

```
int binary(int array[], int n, int v)
{
    int left, right, middle;


    left = -1, right = n;

    while ( left + 1 != right )
    {

        cout << "循环" << endl;
    }

    return -1;
}
```

上面这样写，意味着循环终止的条件是left +1 == right ,由于我们采用的是左开右开表示法，也就是 区间为 (left ,right),如果用个例子来解释，大家就会豁然开朗了， 假如(left=3， right=4)，区间实际上是出现了 右边界=3 ,左边界=4这种情形，也就是右边界 < 左边界，这意味着区间长度<=0 了，无法继续二分了。

第三步: 在第一，二步正确的情况下，写对应的边界更新语句

如果选用的是左闭右开表示法来表示初始化区间:

```
int binary(int array[], int n, int v)
{
    int left, right, middle;

    left = 0, right = n ;

    while (  left < right )
    {
        middle = (left + right) / 2;

        if (array[middle] > v)
        {
            right = middle ;
        }
        else if (array[middle] < v)
        {
            left = middle + 1 ;
        }
        else
        {
            return middle;
        }

        cout << "循环" << endl;
```

```
    }

    return -1;
}
```

为什么left 和right 的更新条件要这样写呢?

```
if (array[middle] > v)， 那么说明key 值肯定在[left , mid-1]中， 但是由于初始区间用了左闭右开表示法，我们应该表示为[left ,mid)
所以right = mid  而不是right = mid -1 ，右开右开!

if (array[middle] < v),那么说明key值肯定在[mid+1,right)中，因为左闭右开表示法,我们应该表示为[mid+1 ,right),所以left = mid +1,
而不是left = mid; 左闭左闭
```

如果选用的是左闭右闭表示法来表示初始化区间:

```
int binary(int array[], int n, int v)
{
    int left, right, middle;

    left = 0, right = n-1 ;

    while (  left <= right )
    {
        middle = (left + right) / 2;

        if (array[middle] > v)
        {
            right = middle - 1;
        }
        else if (array[middle] < v)
        {
            left = middle + 1 ;
        }
        else
        {
            return middle;
        }


        cout << "循环" << endl;
    }

    return -1;
}
```

为什么left 和right 的更新条件要这样写呢?

```
if (array[middle] > v)， 那么说明key 值肯定在[left , mid-1]中，又由于初始区间用了左闭右闭表示法，我们应该表示为[left ,mid-1]
所以right = mid-1  而不是right = mid ，右闭右闭!


if (array[middle] < v),那么说明key值肯定在[mid+1,right]中，因为左闭右开表示法,我们应该表示为[mid+1 ,right],所以left = mid +1,
而不是left = mid; 左闭左闭
```

如果选用的是左开右开表示法:

```
int binary(int array[], int n, int v)
{
    int left, right, middle;

    left = -1, right = n ;

    while (  left + 1 != right )
    {
        middle = (left + right) / 2;

        if (array[middle] > v)
        {
            right = middle ;
        }
        else if (array[middle] < v)
        {
            left = middle  ;
        }
        else
        {
            return middle;
        }


        cout << "循环" << endl;
    }

    return -1;
}
```

为什么left 和right 的更新条件要这样写呢?

```
if (array[middle] > v)， 那么说明key 值肯定在(left , mid-1]中，又由于初始区间用了左开右开表示法，我们应该表示为(left ,mid)
所以right = mid  而不是right = mid -1 ，右开右开!

if (array[middle] < v),那么说明key值肯定在[mid+1,right)中，因为左开右开表示法,我们应该表示为(mid ,right)， 所以left = mid ，
而不是left = mid + 1; 左开左开!
```

若对上面三个步骤还有疑问或者不太理解，没关系， 继续往下看，下面的分析能让你更好的理解

下面讲讲，二分中经常出现的错误

1.【两个边界写错了一个，造成查找错误】先上段有bug的代码，是不是你常用的呢?

```
#include<cstdio>
#include<algorithm>
#include<iostream>
using namespace std;
int s[1000];
int binary(int array[], int n, int v)
{
    int left, right, middle;

    left = 0, right = n;

    while (left < right)
    {
        middle = (left + right) / 2;
        if (array[middle] > v)
        {
            right = middle - 1;
        }
        else if (array[middle] < v)
        {
            left = middle + 1;
        }
        else
        {
            return middle;
        }

        cout << "循环" << endl;
    }

    return -1;
}

int main()
{
    int n  ;
    cin >> n;

    for(int i = 0; i <  n ; i++)
        cin >> s[i];

    int Index = binary(s,n,6);//查找6
    cout << "key值的下标是:" << Index << endl;
    return 0;

}
/*
6
2 3 4 5 6 7
*/
```

大家将我注释里面的样例拿去跑，会发现查找key = 6,时居然返回-1,。

我们来手动模拟一下上面这段**错误代码**的二分过程:

第一次循环时:

left = 0, right = 6, 即[0， 6)

由于s[3] <6，，，，更新下次循环的待查区间为 [4, 6);

第二次循环时:

left = 4 , right = 6, 即[4, 6)

由于 s[5] > 6.....更新下次循环的待查区间为[4,4);（其实[4, 4)对应着区间[4,3],已经不是一个合法区间了，说明区间长度<=0l了，不能再二分减半了，因此应该退出循环）

然后left == right ==4就退出循环了，返回-1.

我们来分析为什么会有bug:

初始化区间的表示法就打算用左闭右开，既[0 , 6), 那么我们的边界更新语句就应该与之对应，而当满足array[middle] > v的条件时, v如果存在的话应该在[left, mid)区间中,但是上诉循环内更新右端点的语句，居然是用

```
if (array[middle] > v)
{
    right = middle - 1;
}
```

这样的话区间被更新为[left, mid-1), 那么 mid-1这个位置的元素就会被忽略了，万一，mid -1这个点的值就是key值呢，所以啊就像上面分析过程的第二次循环，其实区间应该更新为【4, 5)，，不然你更新后的区间就不包含样例中的6了，

博主是勤劳的小蜜蜂，顺手也把right = mid时 **正确的二分**过程也手动模拟吧（正确的程序，在上面就给出了）:

第一次循环时:

left = 0, right = 6, 既[0 6)

由于s[3] <6，，，，更新下次循环的待查区间为 [4, 6);

第二次循环时:

left = 4 , right = 6, 既[4, 6)

由于 s[5] > 6.....更新下次循环的待查区间为[4,5);

第三次循环时:

left = 4 ,right = 5,既 [4, 5)

由于s [4] == 6 ,return 4;

成功查到!

敲黑板！敲黑板：初始区间的表示法在选定后， 边界更新语句应该与之对应，避免漏掉key值，我这边只是举例左闭右开表示法的错误例子，其他区间表示法，若边界写错了一个，也会出错！！！

2：【两个边界写错了2个，造成死循环】

为什么会死循环，死循环说明了区间长度一直没有减少。例如在某次循环后待查询的区间始终是[0 ,1]，但是在该次循环后

区间始终无法减小，始终是保持着是[0 , 1]；

其实死循环的导致，是因为区间的左右边界条件全部都写错了（也可以说是前面第一种错误情况的一种特例，第一种错误情况是只有 左边界或者右边界写错）

给出一个错误的代码，估计读者也会觉得好熟悉啊：

```
#include<cstdio>
#include<algorithm>
#include<iostream>
using namespace std;
int s[1000];
int binary(int array[], int n, int v)
{
    int left, right, middle;

    left = 0, right = n - 1;

    while (left <= right)
    {
        middle = (left + right) / 2;

        if (array[middle] > v)
        {
            right = middle ;
        }
        else if (array[middle] < v)
        {
            left = middle ;
        }
        else
        {
            return middle;
        }

        cout << "循环" << endl;
    }

    return -1;
}

int main()
{
    int n  ;
    cin >> n;

    for(int i = 0; i <  n ; i++)
        cin >> s[i];

    int Index = binary(s,n,1); //查找1

    cout << "key值的下标是:" << Index << endl;

    return 0;
```

```
}
/*
2
0 1
*/
```

大家将我注释里面的样例拿去跑，会发现查找1时，出现了死循环,。

我们来手动模拟一下上面这段**错误代码**的二分过程:

第一次循环时:

left = 0 , right = 1 ， 既 [0 , 1]

s[0] < 1 , 区间更新为 [0 , 1];

第二次循环时:

left = 0, right =1 ,既 [0 ， 1]

s[0] < 1 , 区间还是更新为 [0 , 1];

接下来，， 区间更新都是为[0，1]，长度无法减少，所以出现了死循环。

出现bug原因分析:

初始区间用了左闭右闭，那么边界更新语句应该是left = mid +1 ,right = mid -1，而不是 left = mid ,rigth = mid,,,两个

边界语句都写错了。

订正后的二分过程模拟（正确代码开头有）：

第一次循环时:

left = 0 , right = 1 ， 既 [0 , 1]

s[0] < 1 , 区间更新为 [1 , 1];

第二次循环时:

left = 1 , right =1 , 既 [1, 1]

s[1] == 1，return 1

查找成功!

敲黑板！敲黑板：再说一遍！！初始区间的表示法在选定后， 两条边界更新语句应该与之对应，否则可能会出现死循环.

规避边界条件的错误后，我们还要注意一个地方可能会溢出，这也是不能忽略的:

```
middle = (left + right) / 2;  容易溢出
middle = left + ( right - left ) /2  这样写不容易溢出
假如left 很大，right 很大，此时第一个语句就会溢出，爆int，因为left + right 的结果可能会大于 int 能表示的最大值
```

完善的二分写法：

下面这个完善的二分写法，是<<编程珠玑>>中的代码，第一次看到后，感觉特别熟悉，校队的旺神学长写的二分模板基本跟里面一模一样(旺神学长貌似是自己调试出来的，此处再次膜拜旺神学长，，ORZ，ORZ)

其实这个完善的二分写法就是开头提到的左开右开写法，但是做了一些小修改。

考虑一种特殊情况，如果数组中的元素都相同，那么查找的时候不一定每次都会返回第一个元素的位置，用开头的3种正确代码去查找，肯定返回的是正中间元素的位置.，，因此进行了以下修改，修改后的代码，可以在非递减序列中，查找某个元素第一次出现的位置，不存在返回-1

```
int binary(int array[], int n, int v)
{
    int left, right, middle;

    left = -1, right = n ;

    while (  left + 1 != right )
    {
        middle = (left + right) / 2;


        if (array[middle] >= v)
        {
            right = middle ;
        }
        else
        {
            left = middle  ;
        }


    }

    if( right == n || array[right]!= v)
        return -1;

    return right;
}
```

二分有很多种，我们只需要掌握一种，建议掌握上面那个完善的写法（健壮），然后多刷题，转化为自己的。递减序列只需要改一改，换汤不换药，读者可以自行改进。

转载请注明来处

2020/11/7 wz摘自https://blog.csdn.net/CCSGTC/article/details/80586181