猜分数

每次考试完同学们都喜欢互相打听分数，如果规定某一科目成绩分数范围：[0,100]，现在小明知道自己的成绩，他让你猜他的成绩，如果猜的高了或者低了都会告诉你，用最少的次数猜出他的成绩，你会如何做？

1.最简单的方法当然就是从0开始猜，一直猜到100分

2.其实在我们根本不知道对方水平的条件下，我们每一次的猜测都想尽量将不需要猜的部分去除掉，而又对小明不了解，不知道其水平到底如何，那么我们考虑将分数均分。

猜测过程

1.那么我们假设当猜完50分之后答案是“低了”，那么我们需要在【51,100】分的区间内继续猜小明的分数，

2.同理，我们继续二分，第二次我们猜75分，当回答是“高了”的时候，我们将其分数区域从【51,100】确定到【51,74】；

3.就此继续下去，直到回复是正确为止，这样猜测的次数显然比第一种方式少得多的。

猜测次数

如果顺序猜测0~100，最坏情况下需要101次 顺序查找

而这样每次都猜测分数段中间位置的方式，最坏也只需要约7次

对于更大范围，如n=1000000000，这种方法第一次就能抛弃许多无效状态

基于二分思想的查找方式

二分查找

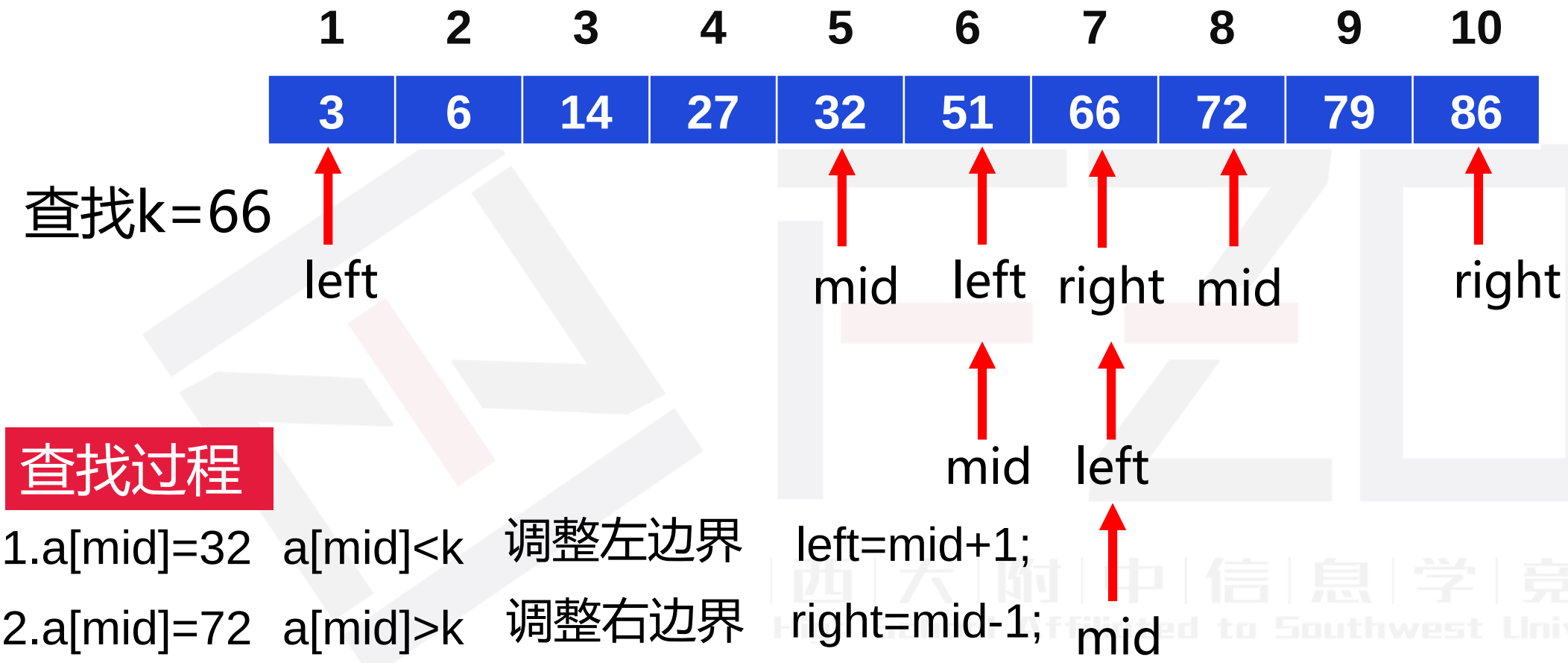
信息学

二分查找

西南大学附属中学校

信息奥赛教练组

查找过程



查找过程

1.a[mid]=32 a[mid]<k 调整左边界 left=mid+1;

2.a[mid]=72 a[mid]>k 调整右边界 right=mid-1;

3.a[mid]=51 a[mid]<k 调整左边界 left=mid-1

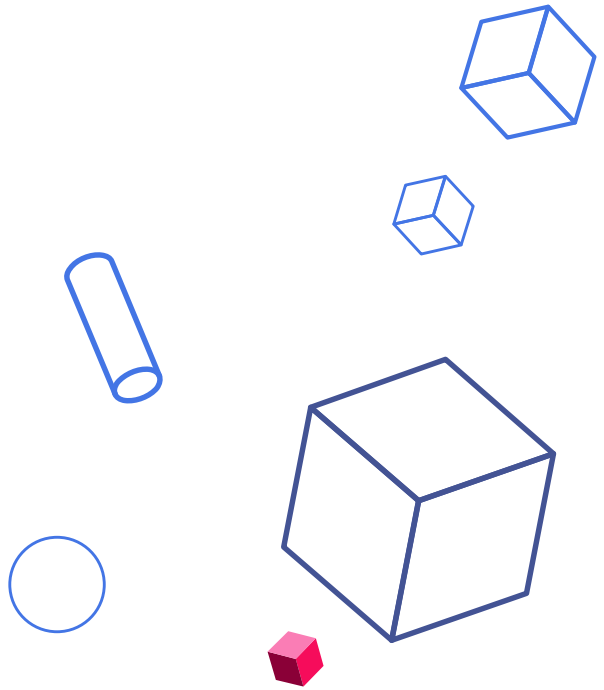
4.a[mid]=66 a[mid]==k 查找成功

二分查找的思想：折半取中，比较

二分查找的前提条件

1.顺序存储(一维数组)
2.单调**有序**

01 基本二分查找



二分查找的核心算法

初始化查找区间[L,R]
计算出mid=(L+R)/2
while(搜索区间长度!=0){

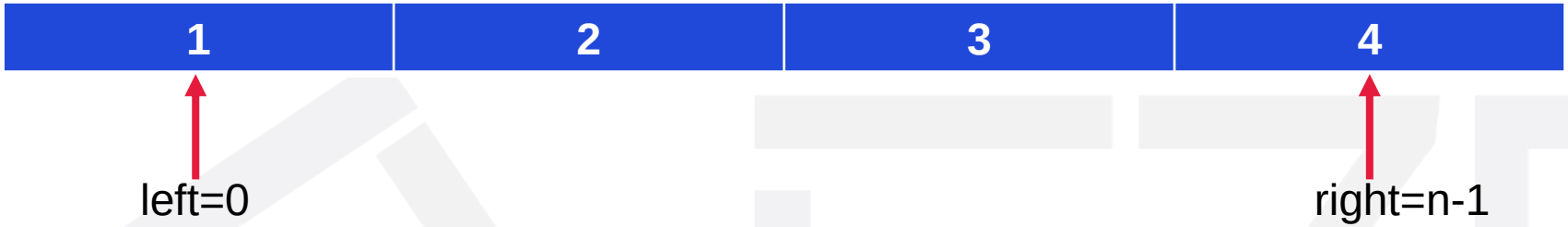
若a[mid]==k,查找成功,停止查找
若a[mid]>k,说明k在左半部分,区间缩小一半
若a[mid]<k,说明k在右半部分,区间缩小一半

}

平均时间复杂度：**O(logn)**

二分查找的两种查找区间

1.左闭右闭[L,R]



2.左闭右开[L,R)

不同的区间状态，二分的本质在于代码逻辑的变化

二分查找参考代码(左闭右闭)


```
left=0,right=n-1;
while ( left <= right ) {
        //int mid = (left + right) / 2;
        int mid = left+(right-left) / 2;  //防止数据溢出
        if (a[mid] == k) {
            ans = mid;
              break;
        }
        else if (a[mid] < k) {
            left = mid + 1;
        }
        else if (a[mid] > k) {
            right = mid - 1;
        }
  }
```

起始区间：[left,right]

循环正常结束：left=right+1

结束区间：[right+1,right]

二分查找参考代码(左闭右开)


```
left=0,right=n;
while ( left < right ) {
        //int mid = (left + right) / 2;
        int mid = left+(right-left) / 2;   //防止数据溢出
        if (a[mid] == k) {
            ans = mid;
             break;
        }
        else if (a[mid] < k) {
            left = mid + 1;
        }
        else if (a[mid] > k) {
            right = mid;
        }
  }
```

起始区间：[left,right)

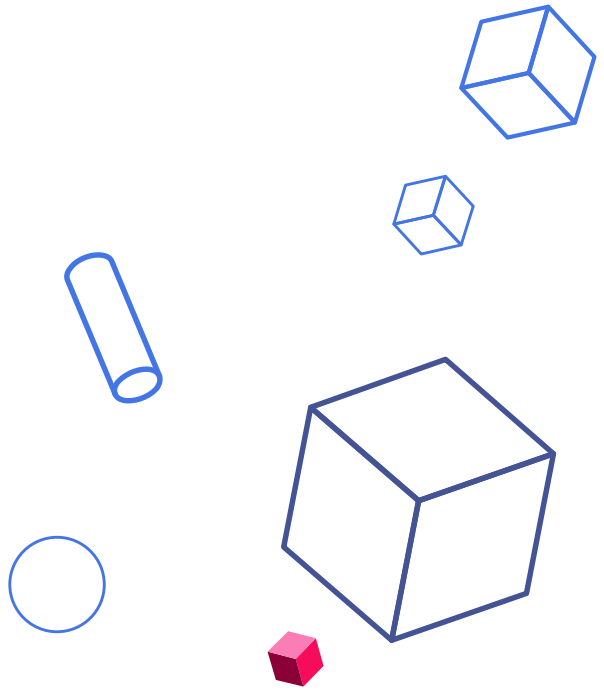
循环正常结束：left=right

结束区间：[right,right)

小结：起始区间，和最终查找结束的区间的开闭状态是一致的

FZOI
西大附中信息学竞赛
High School Affiliated to Southwest University

西南大学附属中学
High School Affiliated to Southwest University



02 二分查找变形

查找边界

有时候会出现这样的情况:

1 2 2 2 4

- 第一个k在什么位置
- 最后一个k在什么位置
- 查找有多少个相同的k

方法一: 找到某一个等于k的位置，然后左右搜寻

数据量大的情况下，时间复杂度高，不能体现二分的高效

方法二: 二分查找到左边界和右边界，边界值相减即可

寻找左边界(双闭区间)


```
int left = 0, right = n- 1;  //左闭右闭
while (left <= right) { // 注意<=
    int mid = left + (right - left) / 2;
    if (a[mid] < k) {
        left = mid + 1;
    }
    else if (a[mid] > k) {
        right = mid - 1;
    }
    else if (a[mid] == k) {
        right = mid - 1; // 锁定左侧边界，缩小右边界
    }
}
// 最后要检查 left 越界的情况
if (left >= n|| a[left] != k)  {k查找失败}
答案输出left的值
```

核心：找到后，不断缩小右边界

寻找右边界(双闭区间)


```
int left = 0, right = n - 1;
while (left <= right) {
    int mid = left + (right - left) / 2;
    if (a[mid] < k) {
        left = mid + 1;
    }
    else if (a[mid] > k) {
        right = mid - 1;
    }
    else if (a[mid] == k) {
        left = mid + 1; // 锁定右侧边界，缩小左边界
    }
}
// 最后要检查 right 越界的情况
if (right < 0 || a[right] != k) {k查找失败}
答案输出right的值
```

核心：找到后，不断缩小左边界

寻找左边界(左闭右开区间)


```
int left = 0,right = n; // 左闭右开
while (left < right) { // 注意<
    int mid = left + (right - left) / 2;
    if (a[mid] > k) {
        right = mid;
    }
    else if (a[mid] < k) {
        left = mid + 1;
    }
    else if (a[mid] == k) {
        right = mid; // 注意
    }
}
// 最后要检查 left 越界的情况
if (left >= n || a[left] != k) {k查找失败}
答案输出left;
```

注意与左闭右开的区别

寻找右边界(左闭右开区间)


```
int left = 0, right = n;
while (left < right) {
    int mid = left + (right - left) / 2;
    if (a[mid] < k) {
        left = mid + 1;
    }
    else if (a[mid] > k) {
        right = mid;
    }
    else if (a[mid] == k) {
        left = mid + 1;
    }
}
// 最后要检查 right 越界的情况
if (right <= 0 || a[right-1] != k) {k查找失败}
```

答案输出right-1;

注意与左闭右开的区别

STL提供二分查找的功能

lower_bound(首地址，尾地址，查找值)：第一个大于或等于k的数字，找到返回该数字的地址

upper_bound(首地址，尾地址，查找值)：第一个大于k的数字，找到返回该数字的地址

找到地址后，减去首地址即得到下标

内部实现方式是**左闭右开**区间

左闭右开是表示区间最友好最出色的方式，原因如下：

1.上下界之差直接表示元素的个数

2.在表示两个相邻子序列时相当方便和简洁：一个子序列的上界就是另一个子序列的下界

3.当上界等于下界时即可表示空集，不会出现上界小于下界的情况。

4.在左闭右开的情况下，能将常规的二分查找（会形成左子序列，中数，右子序列，这实际上是3分查找了）实现为真·二分查找（只形成左子序列和右子序列）

参考博客

- 为什么选择左开右闭?
  https://blog.csdn.net/m0_37302219/article/details/107180126

- 左闭右开和左闭右闭写法中各种小细节深究
  https://www.icode9.com/content-4-853308.html

- 二分查找各类bug总结(注意里面的代码不一定是无bug的，主要看思路)
  https://blog.csdn.net/sunmenggmail/article/details/7540970

Thanks

For Your Watching