关于二分查找的那些事

如果对本文档有任何疑问，欢迎联系`1706zcy@buaa.edu.cn`

0.我对`bsearch`有话说:大人，时代变了!

我们在c语言中采用的 `bsearch` 函数进行二分查找的时候，实例如下。

（该示例引用自 该网页）

```
#include <stdio.h>
#include <stdlib.h>


int cmpfunc(const void * a, const void * b) {
   int* a = (int*)p1, * b = (int*)p2;
    if (*a < *b)return -1;
    else if (*a > *b) return 1;
    else return 0;
}

int values[] = { 5, 20, 29, 32, 63 };

int main ()
{
   int *item;
   int key = 32;

   /* 使用 bsearch() 在数组中查找值 32 */
   item = (int*) bsearch (&key, values, 5, sizeof (int), cmpfunc);
   if( item != NULL )
   {
      printf("Found item = %d\n", *item);
   }
   else
   {
      printf("Item = %d could not be found\n", *item);
   }

   return(0);
}
```

这种二分查找方式

- 在**所有元素两两不相同**的情况下没有问题
- 但是在出现相同元素的情况下，**只能返回其中一个等于查找值的元素**

而在我们课程的几道题目

- 惹 Shy 的 二 分 查 找
- 二荷的二分查找
- 小迷弟搞偷袭

为了得到一个较为精确的答案（而不是其中一个任意解），我们需要的都是找到**第一个大于某数或者大于等于某数**的元素。这个时候光用`bsearch`函数就没什么用了。

但是实际上，如果自己手写二分查找的话，**只要搜索条件正确，同样可以做到和`bsearch`一样，在$O(\log n)$的时间复杂度内完成搜索。**

下面我们来具体介绍这两种二分查找的实现方式，以及如何使用他们完成题目。

1.`lower_bound` 下界二分查找

针对这一个函数，我们要做的是

在一个有序的序列中，找到第一个大于等于某个数的值

注意，这里找到的不一定是严格等于要查找的值，只要是**第一个 大于等于**的就行了

具体的函数如下

```
//下界二分查找
//a: 要查找的有序数组，默认是从小到大排序
//lo, hi: 要查找的范围是从a[lo]到a[hi]之间(包含a[lo]和a[hi])
//val: 要查找的值
//返回的值: 找到的元素在数组a中的下标
//如果所有数都小于val,则返回hi+1
int lower_bound(int a[], int lo, int hi, int val) {
    if (val > a[hi]) return hi + 1;
    int mi = 0;
    while (lo < hi) {
        mi = (lo + hi) >> 1;
        if (a[mi] < val) lo = mi + 1;
        else hi = mi;
    }
    return lo;
}
```

具体例子如下:

假如说我们要查找的有序数列为 $[1,2,2,2,3,5]$ 此时假定数组第一个元素的下标为 1

- 在查找 2 的时候，那么我们此时返回的下标就是 2 (注意：只需要找到第一个)
- 在查找 3 的时候，那么我们此时返回的下标就是 5
- 在查找 4 的时候，那么我们此时返回的下标就是 6

2.`upper_bound` 上界二分查找

针对这一个函数，我们要做的是

在一个有序的序列中，找到第一个严格大于某个数的值

```
//上界二分查找
//a: 要查找的有序数组，默认是从小到大排序
//lo, hi: 要查找的范围是从a[lo]到a[hi]之间(包含a[lo]和a[hi])
//val: 要查找的值
//返回的值: 找到的元素在数组a中的下标
//如果所有数都小于等于val,则返回hi+1
int upper_bound(int a[], int lo, int hi, int val) {
```

```
    if (val >= a[hi]) return hi + 1;
    int mi = 0;
    while (lo < hi) {
        mi = (lo + hi) >> 1;
        if (a[mi] <= val) lo = mi + 1;
        else hi = mi;
    }
    return lo;
}
```

具体例子如下:

假如说我们要查找的有序数列为 $[1,2,2,2,3,5]$ 此时假定数组第一个元素的下标为 1

- 在查找 1 的时候，那么我们此时返回的下标就是 2 (注意：这里要找的是**第一个严格大于的**)
- 在查找 3 的时候，那么我们此时返回的下标就是 6
- 在查找 5 的时候，那么我们此时返回的下标就是 7
- (这个下标不存在，他反应的就是数列里的所有数都不大于 5，返回的是下标，所以也不需要担心数组越界)

3.例题解析

说了这么多，我们还是结合一些题目

C5-惹 Shy 的二分查找

题目大意

- 数组中所有的数已经从小到大有序排列
- 找到一个数在一个有序数组当中**第一次出现的位置**
- 如果该数没有出现，则输出 -1

题目分析

本题很明显，是一个**直接考察 `lower_bound` 应用的题目。**

此时我们可以直接采用该函数，找到第一个**大于等于**查找值的元素的下标

- 如果该下标对应的元素**等于**查找值，则说明该元素就是我们要找的解
- 如果该下标对应的元素**大于**查找值，则说明该元素在本数列当中**没有出现过**

那么具体的代码如下

```
#include<stdio.h>
int lower_bound(int a[], int lo, int hi, int val) {
    if (val > a[hi]) return hi + 1;
    int mi = 0;
    while (lo < hi) {
        mi = (lo + hi) >> 1;
        if (a[mi] < val) lo = mi + 1;
        else hi = mi;
    }
    return lo;
}
int n, m;
int a[1919810], x;
```

```
int pos;
int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; ++i)
        scanf("%d", &a[i]);//从a[1]开始读入 读到a[n]
    while (m--) {
        scanf("%d", &x);
        pos = lower_bound(a, 1, n, x);//下界二分查找的下标 注意起点是1 终点是n
        //如果是从a[0]读到a[n-1] 那么上面的函数就应该传入0和n-1才对
        if (a[pos] == x) printf("%d\n", pos);//相等:我们找到了所求解
        else printf("-1\n");//大于:要查找的值没有出现过
    }
}
```

E8-小迷弟搞偷袭

题目大意

- 数组中所有的数乱序排列
- 找到第一个**严格小于**要查找的数的值，并输出这个值
- 如果所有数都**不小于**要查找的值，则输出 `too weak`

题目分析

本题是一个考察 `qsort` 与 `upper_bound` 的综合应用

`qsort` 不多说了，主要是排序的比较函数需要注意溢出，这个再不会就没救了（

`upper_bound` 是在一个**非降序**的序列中找第一个**严格大于**要查找的数的值

那么我们直接把这题改成在一个**非升序**的序列中找到第一个**严格小于**要查找的数的值就好了呀

（虽然说把所有输入的数取相反数之后，就可以直接套 `upper_bound`，但是 -2147483648 的相反数无法用 `int` 类型进行表示，此时我们就需要用 `long long` 等数据类型表示了）

如何修改查找条件，满足上述条件?

在一个**非降序**的序列中找第一个**严格大于**要查找的数的值的时候，我们的查找条件是

```
while (lo < hi) {
    mi = (lo + hi) >> 1;
    if (a[mi] <= val) lo = mi + 1;
    else hi = mi;
}
```

在一个**非升序**的序列中找到第一个**严格小于**要查找的数的值的时候，我们直接反着求就好了

把**小于等于**改成**大于等于**就行了（注意，**不是改成大于**，这点要尤其注意）

那么在本题当中就是

```
while (lo < hi) {
    mi = (lo + hi) >> 1;
    if (a[mi] >= val) lo = mi + 1;//修改查找条件
    else hi = mi;
}
```

那么这道题目的做法就有了

- 先将数组用`qsort`函数进行**非升序排列**
- 然后采用反过来改造的`upper_bound`函数在其中找到**第一个严格小于**查找数的值
- 如果所有数都大于等于查找值，则输出`too weak`

具体代码如下

```
#include<stdio.h>
#include<stdlib.h>
int cmp(const void* p1, const void* p2) {
    int* a = (int*)p1, * b = (int*)p2;
    if (*a > *b)return -1;//注意cmp函数要进行降序排列
    else if (*a < *b) return 1;
    else return 0;
}
//将"非降序序列中找第一个严格大于的数"改成"非升序序列中找第一个严格小于的数"
int upper_bound(int a[], int lo, int hi, int val) {
    if (val <= a[hi]) return hi + 1;
    int mi = 0;
    while (lo < hi) {
        mi = (lo + hi) >> 1;
        if (a[mi] >= val) lo = mi + 1;
        else hi = mi;
    }
    return lo;
}
int n, m;
int a[114514], x;
int pos;
int main() {
    scanf("%d%d", &n, &m);
    for (int i = 1; i <= n; ++i)scanf("%d", &a[i]);
    qsort(a + 1, n, sizeof(a[0]), cmp);
    while (m--) {
        scanf("%d", &x);
        pos = upper_bound(a, 1, n, x);
        if (pos > n) puts("too weak");//所有数都不小于查找数
        else printf("%d\n", a[pos]);//找到了第一个严格小于查找数的数
    }
}
```

E7-二荷的二分查找与本题要求的几乎一致，还比本题少一步排序，各位可以参考该程序，做出本题。

R1-二分查找PLUS

题目大意

- 给定一个从小到大的有序数列
- 寻找某个数是否出现过
- 如果出现过，输出其**第一次出现的位置**以及**出现次数**
- 没出现过则输出`No`

题目分析

题目的强度较弱，使得一些原本会超时的做法也过了，**此处只讲正解要怎么做**

本题实际上是一个`lower_bound`和`upper_bound`综合应用的题目。

至于找一个数是否出现过，以及第一次出现的位置，我们直接看上面C5-惹 Shy 的 二 分 查 找 这题的解法即可。

至于找到其出现的次数，我们可以利用`upper_bound`，找到第一个**严格大于**查找数的数。

假如在一个非降序的序列当中，第一个**大于等于**查找数的数的下标是 L ，第一个**严格大于** 查找数的数的下标是 R

那么很显然有以下结论

- 下标处于 L 和 $R-1$ 之间（包含 L 和 $R-1$ 这两个下标）的所有数都等于要查找的数。
- 要查找的数出现的次数是 $R-L$
- 当 $R-L=0$ 的时候，就说明**该数没有在本数列中出现过**
- - 这个问题我们回到`lower_bound`和`upper_bound`的本质，第一个大于等于查找数的数也恰好就严格大于查找数，那很显然意味着查找的数没出现过嘛

那么本题要做的就简单了

- 直接调用上述给出的，针对非降序数列的`lower_bound`和`upper_bound`函数
- 首先调用`lower_bound`函数，确保找到的下标对应的数**正好等于**要查找的数，才说明这个数在数列当中
- 然后调用`upper_bound`函数，这两个下标相减就是我们该数出现的个数
- - 直接利用下标相减不等于 0 来判断是否出现过，很显然也是正确的

```
#include<stdio.h>
#define maxn 114514
int n, q;
int a[maxn], b;
int ans;
int lower_bound(int a[], int lo, int hi, int val) {
    if (val > a[hi]) return hi + 1;
    int mi = 0;
    while (lo < hi) {
        mi = (lo + hi) >> 1;
        if (a[mi] < val) lo = mi + 1;
        else hi = mi;
    }
    return lo;
}
int upper_bound(int a[], int lo, int hi, int val) {
    if (val >= a[hi]) return hi + 1;
    int mi = 0;
    while (lo < hi) {
        mi = (lo + hi) >> 1;
        if (a[mi] <= val) lo = mi + 1;
        else hi = mi;
    }
    return lo;
}
int main() {
```

```
    scanf("%d", &n);
    for (int i = 1; i <= n; ++i)//已经确保升序排列
        scanf("%d", &a[i]);
    scanf("%d", &q);
    while (q--) {
        scanf("%d", &b);
        //分别采用下界和上界二分查找
        int l = lower_bound(a, 1, n, b), r = upper_bound(a, 1, n, b);

        //以下两句if条件 选择其中任何一个去写都是对的
        if (a[l] == b) printf("Yes %d %d\n", l, r - l);
        //if (r - l > 0) printf("Yes %d %d\n", l, r - l);
        else puts("No");
    }
}
```

4.如何用bsearch实现lower_bound和upper_bound

该内容引用自 李淳一助教的一篇博客

如果各位对bsearch情有独钟的话，那就试试以下代码吧，在这里不具体介绍了

```
#include<stdio.h>
#include<stdlib.h>
int A[100005];  // 示例全局数组

int compare(const void * a, const void * b) {
    int* a = (int*)p1, * b = (int*)p2;
    if (*a < *b)return -1;
    else if (*a > *b) return 1;
    else return 0;
}

// 查找首个不小于待查元素的元素的地址
int lower(const void *p1, const void *p2) {
    int *a = (int *)p1;
    int *b = (int *)p2;
    if ((b == A || compare(a, b - 1) > 0) && compare(a, b) > 0)
        return 1;
    else if (b != A && compare(a, b - 1) <= 0)
        return -1;  // 用到地址的减法，因此必须指定元素类型
    else
        return 0;
}

// 查找首个大于待查元素的元素的地址
int upper(const void *p1, const void *p2) {
    int *a = (int *)p1;
    int *b = (int *)p2;
    if ((b == A || compare(a, b - 1) >= 0) && compare(a, b) >= 0)
        return 1;
    else if (b != A && compare(a, b - 1) < 0)
        return -1;  // 用到地址的减法，因此必须指定元素类型
    else
        return 0;
}
```

```
int main()
{
    int n;
    scanf("%d",&n);
    int i;
    for(i=0; i<n; i++)
    {
        scanf("%d",&A[i]);
    }
    int k;
    while(scanf("%d",&k)!=EOF)
    {
        int *temp1=bsearch(&k,A,n,sizeof(int),lower);
        if(temp1==NULL)
        {
               temp1=A+n;
        }
        int *temp2=bsearch(&k,A,n,sizeof(int),upper);
        if(temp2==NULL)
        {
               temp2=A+n;
        }
        printf("%d %d\n",temp1-A,temp2-A);
    }
}
```

节后语

没什么好说的，那就祝大家期末顺利，不再害怕二分查找了吧！