

二分

整数二分

二分法是最常用的算法之一。其解决的问题是：

某个搜索区域是一段区间 $I = [l : r]$ ，并且存在一个性质 P ，存在 k 使得任意的 $x \in [l : k]$ ， x 都满足该性质；任意 $x \in [k + 1, r]$ ， x 都不满足性质（当然也可以相反，前半段不满足性质，后半段满足）。我们想求 k 。也就是说，整个搜索区间可以分为两段不相交的部分（这里没说非空，也就是说允许其中一段是空集），使得一段满足某性质，另一段不满足，然后问分界线在哪里。

例如：给定一个从小到大排序的有序数组 a ，求：

1. 大于等于10的第一个数的下标（取性质 P 为 $P(x) \geq 10$ ，那么整个数组左半部分是不满足 P 的，有半部分满足）
2. 小于10的最后一个数的下标（取性质 P 为 $P(x) < 10$ ，同上）

这两个问题都可以用二分来做，找分界线。如果是求小于10的第一个数的下标，那这里就不是求分界线位置了，所以不用二分来做（答案就是 $a[1]$ 而已）。

我们考虑复杂一点的问题，看起来无法用二分，其实可以。给定一个数组 a ，我们想找若将 a 排好序，下标为 k 的数。这个问题当然可以用快速选择来做，并且时间复杂度是 $O(n)$ 的，已经最优。但其实二分也可以做。我们其实要找的是这样的数：设性质 P 为 $P(x)$ 是小于等于 x 的数的个数大于等于 k ，那么答案就是满足 P 的最小的数，搜索范围是 a 里所有数字的范围。比如 a 里所有数字都在 $-10^5 \sim 10^5$ ，那么就可以令搜索范围是 $[-10^5, 10^5]$ 。

还有些问题，不符合上面的思维模板，但我们可以做一些转化使之符合。比如给定一个从小到大排序的有序数组 a ，求 x 在 a 中的下标，如果不存在则返回 -1 。如果取 $P(y) : y = x$ ，这个性质事实上无法将 a 两分；但是我们可以取 $P(y) : y \geq x$ ，这样问题就变为求满足 P 的最左边的数，就能二分了。求得答案之后只需要再验证一下这个数是否等于 x 即可。

例题：https://blog.csdn.net/qq_46105170/article/details/113792973。先看代码，后面讲代码细节

```
1  #include <iostream>
2  using namespace std;
3
4  const int N = 100010;
5  int n, q, k;
6  int a[N];
7
```

```

8  int main() {
9      scanf("%d%d", &n, &q);
10     for (int i = 0; i < n; i++) scanf("%d", &a[i]);
11
12     int x, y;
13     while (q--) {
14         cin >> k;
15         int l = 0, r = n - 1;
16         while (l < r) {
17             int m = l + (r - l >> 1);
18             if (a[m] >= k) r = m;
19             else l = m + 1;
20         }
21
22         if (a[l] != k) {
23             puts("-1 -1");
24             continue;
25         }
26
27         x = l;
28         l = 0, r = n - 1;
29         while (l < r) {
30             int m = l + (r - l + 1 >> 1);
31             if (a[m] <= k) l = m;
32             else r = m - 1;
33         }
34         y = l;
35         printf("%d %d\n", x, y);
36     }
37 }

```

我们把整个二分的代码流程过一遍：

1. 确定搜索区间，保证如果答案存在的话，这个答案一定在搜索区间里。注意，搜索区间要取成闭区间。另外要取一个性质 P ，使得整个区间可以划分为左右两个不相交的部分，并且一部分满足性质，另一部分不满足。两部分可以有空集。并且我们要找的答案恰好在分界线上；
2. 如果搜索区间为空，则直接返回，否则令 l, r 分别为区间左右端点；
3. 接下来是 `while` 循环，要写成 `while(l < r)`
4. 接下来算中点，注意中点有两种形式，`int mid = l + (r - l >> 1)` 和 `int mid = l + (r - l + 1 >> 1)`，具体取那种要看下面；
5. 接下来根据中点是否满足性质，来移动区间端点。我们把“满足性质”标记为 `o`，不满足标记为 `x`，我们举几个例子：
 1. 如果整个区间是 `ooooooxxxxx` 这种样子的，并且我们要找最后一个 `o`，那就应该写：

```

1 | if (P(a[mid])) l = mid;
2 | else r = mid - 1;

```

因为如果中点满足性质，而我们要找最后一个满足性质的位置，从而我们需要挪左端点；否则中点不满足性质，我们就要挪右端点；

2. 如果整个区间是 `xxxx0000000` 这种样子的，并且我们要找第一个 `0`，那就应该写：

```

1 | if (P(a[mid])) r = mid;
2 | else l = mid + 1;

```

道理类似；

3. 如果整个区间是 `000000xxxx` 这种样子的，并且我们要找第一个 `x`，那就应该写：

```

1 | if (!P(a[mid])) r = mid;
2 | else l = mid + 1;

```

如果中点不满足性质，但我们要找第一个不满足的，所以要让右端点向左挪到中点；否则挪左端点。

上面只举了部分例子，但读者此时无论如何都可以将 `if` 语句给写出来了。

6. 接下来根据 `if` 里是写了 `l = mid` 还是 `l = mid + 1` 来调整 `int mid = l + (r - l >> 1)` 和 `int mid = l + (r - l + 1 >> 1)` 这两句话：

1. 如果写了 `l = mid`，那么 `int mid = l + (r - l + 1 >> 1)`
2. 如果写了 `l = mid + 1`，那么 `int mid = l + (r - l >> 1)`

很好记忆，可以这么记，`l` 和 `mid` 一定有且只有一个地方要 `+1`。这是为了防止死循环。读者可以自行尝试。

7. `while` 循环结束，接下来出循环，出循环的时候我们可以保证只有一个候选答案了（因为出循环意味着 `l = r`），现在要判断一下 `a[l]` 是否是我们要找的答案。如果题目中已经能保证答案存在，或者我们可以人为逻辑判断答案一定存在，那可以直接返回 `a[l]`，否则还需要用 `P` 来验证一下 `a[l]` 是否满足。

我们看一道例题，将整个流程完整走一遍：

给定从小到大的有序数组 a ，求小于等于 x 的最后一个数的位置。如果答案不存在，则返回 -1 。性质可以定义为“小于等于 x ”，代码这样写：

```

1 | // 第1步，确定搜索区间，要取闭区间
2 | int l = 1, r = n;
3 | // 第2步，如果搜索区间为空，则直接返回
4 | if (l > r) return -1;
5 | // 第3步，while循环，要写<
6 | while (l < r) {
7 |     // 第4步，写mid
8 |     int mid = l + (r - l + 1 >> 1);
9 |     // 第5步，这是00000xxxxx型的，根据性质挪左右端点。

```

```
10 // 第6步，发现这里是l = mid，调整mid那一行
11 if (a[mid] <= x) l = mid;
12 else r = mid - 1;
13 }
14 // 第7步，判断一下答案一定存在吗？题目没说小于等于x的数一定存在，所以这里还需要判断一下
15 if (a[l] <= x) return l;
16 return -1;
```

二分是一个需要多加练习的算法，练多了才能有深刻的体会，光看理论容易云里雾里。希望读者多加练习。

浮点数二分

浮点数二分的意思是，搜索答案是浮点数。例如对于一个单调实值函数想求其零点，也可以用二分来做。浮点数二分没有整数二分那么多的边界条件需要考虑，简单很多。

例题https://blog.csdn.net/qg_46105170/article/details/113792995。这道题是求 n 的三次方根，由于三次方根是个单调函数，所以可以用二分来做。我们只需要根据题目要求的精度确定一个小数 `eps`，当搜索区间范围小于 `eps` 的时候就退出循环，此时答案就到了需要的精度了。代码如下：

```
1  #include <iostream>
2  using namespace std;
3
4  // 这里精度需要比保留的小数位数多2，题中是保留6位小数，所以此处是1e-8
5  const double eps = 1e-8;
6
7  int main() {
8      double n;
9      cin >> n;
10
11     double l = -50, r = 50;
12     while (l + eps < r) {
13         double m = l + (r - l) / 2.0;
14         if (m * m * m > n) r = m;
15         else l = m;
16     }
17
18     printf("%.61f", l);
19 }
```

作业

上面例题

Leetcode 33 34 35 69 153 278 410 658 704 875