二分查找和二分答案

dbywsc

2025/7

目录

二分查找 – 问题引入

给定一个长度为 10^5 的 **单调递增** 数列 $\{a_n\}$，接下来 $q(1 \le q \le 10000)$ 次询问，每次输入一个数字 t，要求输出第一个大于等于 t 的数的位置，如果不存在这样的数字则输出 -1。

可以想到每次询问时都遍历一次数组 a，但是这样下来的总时间复杂度是 $O(nq)$，无法在短时间内解决问题。

二分查找 – 思路

由于 $\{a_n\}$ 是单调递增的，可以发现下面的性质：
对于 $\{a_n\}$ 中的某一项 a_i，如果 $a_i < t$，那么 a_i 之前的每一项也必然 $< t$；
反之，如果 $a_i \geq t$，那么 a_i 之后的每一项也必然 $\geq t$。
因此，我们可以维护两个变量 l, r 表示查找范围的左右端点，设 $mid = (l + r)/2$，如果 $a_{mid} < t$，那么将 l 设置为 $mid + 1$，继续向右查找；否则将 r 设置为 $mid - 1$，取当前的 mid 暂存为答案，继续向左查找。
如果 l 和 r 重合，那么停止搜索。
在上面的过程中，由于我们每次都截取了当前区间长度的一半，因此总耗费时间为 $O(\log n)$。这种方法就叫做二分搜索 (Binary Search)。

二分查找 – 代码

下面给出解决本问题的代码：

```
int l = 1, r = n;
int ans = -1;
while(l <= r) {
    int mid = (l + r) / 2;
    if(a[mid] < t) {
        l = mid + 1;
    } else {
        r = mid - 1;
        ans = mid;
    }
}
std::cout << ans << std::endl;
```

* 上面的写法实现了闭区间的取法，即可以搜索到 $[1, n]$ 中的每一个位置。在日后我们还会见到不同的写法。它们有些和上面的一样都是闭区间，有些则是半开区间（即 $[1, n)$ 或者 $(1, n]$），或者是开区间（即 $(1, n)$），根据不同的情况我们要选取不同的写法，但是总的来说，闭区间的写法是最常见的。

二分查找 – 例题

单纯的二分查找往往是用来优化代码时间复杂度的技巧或者作为别的算法中的一部分，属于算法竞赛的基本功，在这里列出一道仅需要二分查找就能解决的简单题目用以练习：

P2249 【深基 13. 例 1】查找

一定要注意，能够使用二分查找的前提是**待查找的数组一定是有序的**。

STL 中的二分

事实上，对于二分查找这种常用的操作，在 STL 库中存在封装好了的操作，它们分别是 *lower_bound*() 和 *upper_bound*() 。
用法如下：
lower_bound(a.begin(), a.end(), x)：返回 a 中第一个**大于等于** x 的元素的位置的迭代器，如果不存在这样的元素，那么返回 **尾迭代器** 。
upper_bound(a.begin(), a.end(), x)：返回 a 中第一个**大于** x 的元素的位置的迭代器，如果不存在这样的元素，那么返回 **尾迭代器** 。
不妨使用上面的内容解决这道题：P1102 A-B 数对

二分答案 – 引入

有时，我们可以利用二分的性质，去寻找一个问题的合适的答案。具体来说，如果我们发现一个问题的答案具有 **单调性** ，比如“最大化最小值”或者“最小化最大值”，此时我们就可以利用二分查找答案可能存在的区间，如果当前枚举到的值确实是一个合法的答案，那么我们就将其保存下来并且移动左右端点。

一般来说，二分答案的模版如下：

```
int l = 1, r = n;
int ans = -1;
while(l <= r) {
    int mid = (l + r) / 2;
    if(check(mid)) {
        l = mid + 1;
    } else {
        r = mid - 1;
        ans = mid;
    }
}
std::cout << ans << std::endl;
```

二分答案 – 一道例题 I

P1873 [COCI 20112012 #5] EKO 砍树
可以发现，我们需要尽可能的让 H 更大，满足单调性，因此可以使用二分答案。

二分答案 – 一道例题 II

将树木的高度存为 $\{a_n\}$，答案记做 ans。
首先确定合理的左右端点，当 $H = 0$ 时，显然得到的木材是最多的；当 $H = \max(\{a_n\})$ 是，我们就砍不到任何的木材了。因此，一开始的左右端点应该设置成 $l = 0$，$r = \max(\{a_n\})$。
然后考虑 $check()$ 怎么写，很明显，我们找到当前的高度后，应该拿这个高度和 n 棵树的高度一一比较，记当前高度可以砍到的树木长度总和为 cnt，当前枚举的高度为 x，那么 $cnt = \sum_{i=1}^{n} \max((a_i - x), 0)$。
如果最后 $cnt \geq m$，那么说明当前的高度是一个合适的高度，保存为 ans，然后我们尝试寻找有没有可能有更大的值，因此移动左端点；反之移动右端点。
由于每次 check 时做了一次 $\{a_n\}$ 的遍历，因此总时间复杂度为 $O(n \log n)$。

二分答案 – 一道例题 III

```cpp
bool check(int x) {
    i64 cnt = 0;
    for(int i = 1; i <= n; i++) cnt += std::max(a[i] - x, 0);
    return cnt >= m;
}
int main(void) {
    std::cin >> n >> m;
    for(int i = 1; i <= n; i++) {
        std::cin >> a[i];
        maxn = std::max(maxn, a[i]);
    }
    int l = 1, r = maxn;
    while(l <= r) {
        int mid = (l + r) / 2;
        if(check(mid)) ans = mid, l = mid + 1;
        else r = mid - 1;
    }
    std::cout << ans << std::endl;
    return 0;
}
```

实数二分 – 引入

P1577 切绳子

二分的范围有时不仅适用于整数中，还可以推广到实数域中。比如在这道题中，绳子的长度是任意正实数，因此我们也需要在实数域内进行二分答案。

与整数二分不同的是，当涉及到实数时，进行 $(l+r)/2$ 的操作并不会很容易的让 $l=r$，而是让它们之间的数值不断逼近。因此实数二分的模版并不需要 $l=r$ 才退出循环，而是让 l 和 r 之间的误差小于某个特定的精度即可，这个精度因题而异。

实数二分 – 模版

```
while(r - l >= eps) {
    double mid = (l + r) / 2;
    if(check(mid)) {
        l = mid; ans = mid;
    } else r = mid;
}
```

其中，*eps* 即为我们设置的精度。
同样是因为精度问题，每当涉及实数二分是，题目往往不会要求我们给出非常确切的答案，而是要求给出的答案与标准答案的误差在一定范围内即可。

二分答案是算法竞赛中极其常用的技巧，它作为一种寻找答案的方式，经常用来配合其他算法解决问题，大家一定要熟练掌握。

下面给出本节的练习题：

P2440 木材加工

P2678 [NOIP 2015 提高组] 跳石头

P3853 [TJOI2007] 路标设置

P1182 数列分段 Section II

P3743 小鸟的设备

P10909 [蓝桥杯 2024 国 B] 立定跳远