二分查找：不只是查找值

原创 张忠诚 LOA算法学习笔记 2021-03-03 20:40

提到二分的用途，我们通常是用它在一个有序数组里进行值的查找，但二分的用途却远远不止于此。一些最大值最小化，和最小值最大化的问题也通常是使用二分查找的思想进行求解，下面通过这两个题目看一下二分查找在这类问题上的应用。

01 最小值最大化：两球之前的磁力[1]

1.1 题目描述

在代号为 C-137 的地球上，Rick发现如果他将两个球放在他新发明的篮子里，它们之间会形成特殊形式的磁力。Rick有 n 个空的篮子，第 i 个篮子的位置在 position[i] ，Morty 想把 m 个球放到这些篮子里，使得任意两球间最小磁力最大。已知两个球如果分别位于 x 和 y ，那么它们之间的磁力为 |x - y| 。给你一个整数数组 position 和一个整数 m ，请你返回最大化的最小磁力。

1.2 问题分析

首先给定一个磁力，我们都可以在线性时间内判定是否存在一种放置策略使得任一小球间的磁力最小值不小于给定值，具体判定方法为，对所有篮子的位置从小到大排序，贪心的把第一个球放在第一个篮子中，之后以当前磁力大小为间距大小等间距的依次向后放球，记录在此间距的条件下最多能在篮子里放多少小球，如果可放的小球数大于等于m，则说明我们存在一种放置策略使得任意小球间磁力的最小值不小于给定值。

其次在题目所给的条件下，我们可以求出最小磁力的最小可能取值low，以及最小磁力的最大可能取值high，最小磁力最小可能值显然为1，因为一个篮子不能放两个球。最小磁力的最大可能取值跟m有关，m个球共有m-1个间隔，要想使间隔距离的最小值尽可能的大，就要让每个间隔是等分的，所以最小磁力的最大可能值为(position.back()-position.front())/(m-1)。

最后，我们就可以在最小磁力的最小可能取值low到最小磁力的最大可能取值high的区间内进行二分查找，当mid“合法”时，我们就记录下来，但不能立马返回，因为还可能存在更大的值仍然“合法”，所以我们就需要去区间[mid+1,high]中继续寻找，如果mid不“合法”,说明最小磁力的最大值要比当前值要小，所以我们需要去区间[low, mid - 1]中继续寻找。其中，“合法”意味着存在一种放置策略使得任意小球间磁力的最小值大于等于mid，判断方法参考该小节一开始的贪心策略。

至于为什么合法带引号，则是因为我们的判断策略并不是真的合法，他无法保证每次二分的res都是某两个篮子之间的距离，我们只能保证最后返回的res是真正合法的，这里可以利用反证法证明下，假设最后返回的res不是真正合法的，由于我们本小节第一段的算法保证了任意小球间的磁力不小于res，既然res不是真正合法的，就说明对于所有的i、j，position[j]-position[i]都大于res，那么res+1也必然满足条件，我们的算法就不可能返回res，所以我们最后返回的res是真正合法的。

1.3 代码实现

```
bool check(int mid, vector<int>& position, int m) {
        int count = 1;
        int pre = position[0];
        for (int i = 1; i < position.size(); ++i) {
            if (position[i] >= pre + mid) {
                ++count;
                pre = position[i];
            }
```

```
        }
        if (count >= m) {
            return true;
        } else {
            return false;
        }
    }
    int maxDistance(vector<int>& position, int m) {
        sort(position.begin(), position.end());
        int low = 1;
        int high = (position.back() - position.front()) / (m - 1);
        int res = 1;
        while (low <= high) {
            int mid = low + ((high - low) >> 1);
            if (check(mid, position, m)) {
                res = mid;
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }
        return res;
}
```

1.4 时间复杂度分析

单独二分查找的时间复杂度为O(log(high - low))，其中high=(position.back() - position.front()) / (m - 1)，low = 1。check函数的时间复杂度是O(n)，n为数组的长度。由于每次查找都需要调用一次check函数，所以二分查找整体的时间复杂度为O(nlog(high - low))，此外一开始对数组排序的时间复杂度为O(nlogn)，所以该算法总的时间复杂度为max(O(nlogn), O(nlog(high - low)))。

02 最大值最小化：分割数组的最大值[2]

2.1 题目描述

给定一个非负整数数组和一个整数 m，你需要将这个数组分成 m 个非空的连续子数组。设计一个算法使得这 m 个子数组各自和的最大值最小。

2.2 问题分析

我们仍然可以采用上一题中的三段式的分析方法，首先给定一个值，我们总可以在线性时间内判断是否存在一个划分使所有的连续子数组之和不超过该值。具体判断方法为，定义一个count变量和一个求和变量sum，遍历数组求当前子数组的和，当当前子数组的和大于给定值时，count加1，令sum等于当前数组元素，继续向后遍历，遍历结束后判断count是否小于等于m就可以了。

其次在题目所给条件下，我们可以求出可行解的范围，即子数组各自和最大值的最小可能取值low和子数组各自和最大值的最大可能取值high。这题的low和high比较显然，分别是当前数组的最大值，以及当前数组的和。

最后，我们就可以在low和high的区间内进行二分查找，当mid“合法”时，我们就记录下该值，但不能立即返回，因为可能存在更小的值仍然“合法”，需要去区间[low, mid -1]继续寻找。当mid不“合法”时，说明当前的值太小了，我们需要去区间[mid + 1, high]中继续寻找。其中“合法”的含义为是否存在一个划分使所有连续子数组的和不超过mid，具体判断方法参考本小节的第一段。

这里合法带引号的原因跟第一题的原因是一样的，我们也并不能保证每次二分的res是真正合法的，我们只能保证最后返回的res是真正合法的，即最后返回的res是数组某个连续子数组的和。这里我们也用反证法证明一下，假设最后返回的res不是真正合法的，由于本节第一段的算法保证了所有连续子数组之和都不超过res，如果res不是真正合法的，意味着它不是任意子数组之和，那么任意子数组的和都要小于res，这时候显然res-1也是符合条件的，我们的算法不应该返回res，所以最后返回的res是真正合法的。

2.3 代码实现

```
bool check(int mid, vector<int>& nums, int m) {
        int count = 1;
        int sum = 0;
        for (int i =0; i < nums.size(); ++i) {
            sum += nums[i];
            if (sum > mid) {
                ++count;
                sum = nums[i];
            }
        }
        if (count <= m) {
            return true;
        } else {
            return false;
        }
    }
    int splitArray(vector<int>& nums, int m) {
        int low = -1, high = 0;
        for (int i = 0; i < nums.size() ; ++i) {
            if (low < nums[i]) {
                low = nums[i];
            }
            high += nums[i];
        }
        int res = low;
        while(low <= high) {
            int mid = low + ((high - low) >> 1);
            if (check(mid, nums, m)) {
                res = mid;
                high = mid - 1;
            } else {
                low = mid + 1;
            }
        }
        return res;
    }
```

2.4 时间复杂度分析

单独二分查找的时间复杂度为O(log(sum - max))，其中sum是数组的和，max是数组的最大值，check函数的时间复杂度是O(n)，n为数组的长度，由于每次查找都需要调用一次check函数，所以算法整体的时间复杂度为O(nlog(sum - max))。

03 总结

遇到求最大值最小化，最小值最大化的问题我们都可以尝试通过二分查找的方式来解决，分析问题的时候就可以采取上面的三段式：

1. 首先看能不能找到一个判断合法的算法，所谓合法就是用来告诉我们当前查找的值是大了还是小了，从而缩小查找范围，该方法要尽可能的简单，通常可以采取贪心的思想。
2. 其次确定可能解的范围，这个取值范围有多种可能，比如第一题最小磁力的最大可能取值我们也可以不除m-1，但这样查找范围扩大，算法性能显然会下降，所以我们的范围应该尽可能的小。
3. 最后在可能解的范围内进行二分查找就可以了，这里写法很固定，套上二分的模版就行，最大值最小化和最小值最大化正好是对称的。

04 题目链接

这两个题目都是取自leetcode，大家看完后可以去leetcode写写代码试一下，以下是题目链接。
[1]https://leetcode-cn.com/problems/magnetic-force-between-two-balls/
[2]https://leetcode-cn.com/problems/split-array-largest-sum/