题目地址（796. Minimum Light Radius）

https://binarysearch.com/problems/Minimum-Light-Radius

入选理由

1. 能力检测 + 最左二分的典型题目

标签

- 二分

难度

- 困难

题目描述

```
You are given a list of integers nums representing coordinates of houses on a 1-dimensional line. You have 3 street ligh

Constraints

n ≤ 100,000 where n is the length of nums
Example 1
Input
nums = [3, 4, 5, 6]
Output
0.5
Explanation
If we place the lamps on 3.5, 4.5 and 5.5 then with r = 0.5 we can light up all 4 houses.
```

前置知识

- 排序
- 二分法

二分法

思路

本题和力扣 **475. 供暖器** 类似。

这道题含义是给你一个数组 nums，让你在 [min(nums),max(nums)] 范围内放置 3 个灯，每个灯覆盖范围都是 r，让你求最小的 r。

之所以不选择小于 min(nums) 的位置和大于 max(nums) 的位置是因为没有必要。比如选取了小于 min(nums) 的位置 pos，那么选取 pos **一定不比选择 min(nums) 位置好**。

这道题的核心点还是一样的思维模型，即：

- 确定 r 的上下界，这里 r 的下界是 0 上界是 max(nums) - min(nums)。
- 对于上下界之间的所有可能 x 进行枚举（不妨从小到大枚举），检查半径为 x 是否可以覆盖所有，返回第一个可以覆盖所有的 x 即可。

注意到我们是在一个有序序列进行枚举，因此使用二分就应该想到。可使用二分的核心点在于：如果 x 不行，那么小于 x 的所有半径都必然不行。

接下来的问题就是给定一个半径 x，判断其是否可覆盖所有的房子。

这其实就是我们讲义中提到的**能力检测二分**，我定义的函数 possible 就是能力检测。

首先**对 nums 进行排序**，这在后面会用到。 然后从左开始模拟放置灯。先在 nums[0] + r 处放置一个灯，其可以覆盖 [0, 2 * r]。由于 nums 已经排好序了，那么这个灯可以覆盖到的房间其实就是 nums 中坐标小于等于 2 * r 所有房间，使用二分查找即可。对于 nums 右侧的所有的房间我们需要继续放置灯，采用同样的方式即可。

能力检测核心代码：

```
def possible(diameter):
    start = nums[0]
    end = start + diameter
    for i in range(LIGHTS):
        idx = bisect_right(nums, end)
        if idx >= N:
            return True
        start = nums[idx]
        end = start + diameter
    return False
```

由于我们想要找到满足条件的最小值，因此可直接套用**最左二分模板**。

代码

代码支持：Python3, Java，CPP

Python3 Code:

```python
class Solution:
    def solve(self, nums):
        nums.sort()
        N = len(nums)
        if N <= 3:
            return 0
        LIGHTS = 3
        # 这里使用的是直径，因此最终返回需要除以 2
        def possible(diameter):
            start = nums[0]
            end = start + diameter
            for i in range(LIGHTS):
                idx = bisect_right(nums, end)
                if idx >= N:
                    return True
                start = nums[idx]
                end = start + diameter
            return False

        l, r = 0, nums[-1] - nums[0]
        while l <= r:
            mid = (l + r) // 2
            if possible(mid):
                r = mid - 1
            else:
                l = mid + 1
        return l / 2
```

Java Code:

```java
class Solution {
    public double solve(int[] nums) {
        Arrays.sort(nums);
        int streetLength = nums[nums.length - 1] - nums[0];
        int low = 0, high = streetLength / 3 + 1;
        while (low + 1 < high) {
            int mid = low + (high - low) / 2;
            if (isPossible(nums, mid, 3)) {
                high = mid;
            } else {
                low = mid;
            }
        }
        if (isPossible(nums, low, 3)) {
```

```
            return low / 2D;
        }
        return high / 2D;
    }

    private boolean isPossible(int[] nums, int diameter, int lightNumber) {
        int lightDiameter = -1;
        int currentLightNum = 0;
        for (int i = 0; i < nums.length; i++) {
            if (nums[i] > lightDiameter) {
                currentLightNum++;
                lightDiameter = nums[i] + diameter;
            }
            if (currentLightNum > lightNumber) {
                return false;
            }
        }
        return true;
    }
}
```

CPP Code:

```
bool isPossible(vector<int>& nums, int mid) {
    int start = nums[0];
    int end = start + mid;
    for (int i = 0; i < 3; i++) {
        int index = 0;
        while (nums[index] <= end) index++;
        if (index >= nums.size()) return true;
        start = nums[index];
        end = start + mid;
    }
    return false;
}

double solve(vector<int>& nums) {
    if (nums.size() <= 3) return 0;
    sort(nums.begin(), nums.end());
    int l = 0;
    int r = nums[nums.size() - 1] - nums[0];
    while (l <= r) {
        int mid = l + (r - l) / 2;
        if (isPossible(nums, mid)) {
            r = mid - 1;
        } else {
            l = mid + 1;
        }
    }
```

```
    return (double)l / 2;
}
```

复杂度分析

令 n 为数组长度。

- 时间复杂度：possible 复杂度为 logn，主函数复杂度为 $\log(\mathrm{MAX}-\mathrm{MIN})$。其中 MAX 数组为最大值，MIN 为数组最小值。除此之外，还进行了排序， 因此时间复杂度大约是 $\mathrm{O}(\mathrm{nlogn}+\mathrm{logn}*\log(\mathrm{MAX}-\mathrm{MIN}))$。
- 空间复杂度：取决于排序的空间消耗

力扣的小伙伴可以**关注我**，这样就会第一时间收到我的动态啦~

以上就是本文的全部内容了。大家对此有何看法，欢迎给我留言，我有时间都会一一查看回答。更多算法套路可以访问我的 LeetCode 题解仓库：https://github.com/azl397985856/leetcode 。 目前已经 40K star 啦。大家也可以关注我的公众号《力扣加加》带你啃下算法这块硬骨头。

上一页 下一页