# Leetcode 1095 山脉数组中查找目标值

#leedcode

## 题目解析

> 给你一个 **山脉数组** mountainArr，找到数组中等于目标值的 **最小下标值**。

何为**山脉数组**？如果数组 A 是一个山脉数组的话，那它满足如下条件：
首先，A.length >= 3
其次，在 0 < i < A.length - 1 条件下，存在 i 使得：
A[0] < A[1] < ... A[i-1] < A[i]
A[i] > A[i+1] > ... > A[A.length - 1]

你将**不能直接访问该山脉数组**，必须**通过MountainArray接口**来获取数据：

- **MountainArray.get(k)** - 会返回数组中索引为k的元素（下标从 0 开始）
- **MountainArray.length()** - 会返回该数组的长度

注意：对 MountainArray.get 发起超过 **100 次** 调用的提交将被视为错误答案 ==》 无法遍历整个数组

输入的范围：

- 3 <= mountain_arr.length() <= 10000
- 0 <= target <= 10^9
- 0 <= mountain_arr.get(index) <= 10^9

## 二分法题解

### 二分法查找山脉峰值

使用二分法查找的关键在于，待查找序列为**单调**的。
山脉数组的峰值左边为单调递增序列，峰值右边为单调递减序列，所以要先找到山脉数组的峰值。

对于一个范围 [i, j]，我们可以先找到范围 [i, j] 中间连续的两个点 mid 与 mid + 1。

- 如果 mountainArr.get(mid + 1) > mountainArr.get(mid)，那么可以知道峰值在范围 [mid + 1, j] 内
- 如果 mountainArr.get(mid + 1) < mountainArr.get(mid)，那么可以知道峰值在范围 [i, mid] 内
- 通过这样的方法，我们可以在O(logn) 的时间内找到峰值所处的下标

可以利用这一思想解决 LeetCode 852题 [852. 山脉数组的峰顶索引]

```
l, r, = 0, mountain_arr.length() - 1
while l < r:
    mid = (l + r) // 2
    if mountain_arr.get(mid) < mountain_arr.get(mid + 1):
        l = mid + 1
    else:
        r = mid

# 最后的 l 和 r 相等，均为峰值
max_mountain_arr_value = l
```

## 峰值两侧分别进行二分查找

先在峰值左边递增序列使用二分法寻找目标值：

```
l, r = 0, max_mountain_arr_value
while l <= r:
    mid = (l + r) // 2
    cur = mountain_arr.get(mid)
    if cur == target:
        return mid
    elif cur < target:
        l = mid + 1
    else:
        r = mid - 1
```

若峰值左边没有找到，继续在峰值右边递减序列进行查找目标值：

```
l, r = max_mountain_arr_value + 1, mountain_arr.length() - 1
while l <= r:
    mid = (l + r) // 2
    cur = mountain_arr.get(mid)
    if cur == target:
        return mid
    elif cur < target:
        r = mid - 1
    else:
        l = mid + 1
```

若都没有找到，返回 -1

```
return -1
```

为简化代码，峰值左右两边的二分查找可以合并成一个函数：

```
def binary_search(target, l, r, key=lambda x: x):
    target = key(target)
    while l <= r:
        mid = (l + r) // 2
        cur = key(mountain_arr.get(mid))
        if cur == target:
            return mid
        elif cur < target:
            l = mid + 1
        else:
            r = mid - 1

    return -1
```

峰值左侧序列查找：

```
index = binary_search(target, 0, max_mountain_arr_value)
```

如果左侧序列没有找到，峰值右侧序列查找：

```
if index != -1:
    return index
else:
    return binary_search(target, max_mountain_arr_value + 1,
mountain_arr.length() - 1, lambda x: -x)
```

## 山脉数组调用接口实现

本地调试时，需要实现山脉数组的调用代码：

```
class MountainArray:
    def __init__(self, value_list):
        self.value_list = value_list

    def get(self, index: int) -> int:
        return self.value_list[index]

    def length(self) -> int:
        return len(self.value_list)
```

## 复杂度分析

时间复杂度：**O(logn)**，我们进行了三次二分搜索，每次的时间复杂度都为O(logn)

空间复杂度：**O(1)**，只需要常数的空间存放若干变量