20 最接近的三数之和

更新时间：2019-08-30 10:07:08

> 横眉冷对千夫指，俯首甘为孺子牛。
>
> ——鲁迅

刷题内容

> 难度: Medium
>
> 原题链接：https://leetcode-cn.com/problems/3sum-closest/

内容描述

给定一个包括 n 个整数的数组 nums 和 一个目标值 target。找出 nums 中的三个整数，使得它们的和与 target 最接近。返回这三个数的和。假定每组输入只存在唯一答案。

例如：

给定数组 nums = [-1, 2, 1, -4], 和 target = 1.

与 target 最接近的三个数和为 2. (-1 + 2 + 1 = 2).

解题方案

思路 1：时间复杂度: O(N^2) 空间复杂度: O(1)

这道题基本和3.6小节一样，所以解法思路相差不大。同样是固定一个元素，设定两个左右边界指针来进行循环。但是有一点需要注意：这道题最后的结果是一个数值，所以不需要处理重复元素。

因为这道题最后要我们返回的最接近的那个和，而不是所有的组合，因此我们不需要去管重复元素，所以这里我们可以相对于3.6小节放聪明些，在这里我不对思路做具体详述了，差别不大，简单说下代码的执行流程吧：

- 先把 `nums` 进行升序排序，这一步不用说；
- 获取左右边界索引，左边界 l ，右边界 r ；
- `nums[i]` 固定， `while` 循环使左右边界向中间移动；
- `temp` = `nums[i]` + `nums[l]` + `nums[r]` 判断 temp 和 target 是否相等，相等直接返回 target，如果大了则右边界 r 向左移动，如果小了，左边界 l 向右移动。

下面来看下具体代码实现：

Python beats 86.89%

这里用了 `float('inf')` 这个定义，这就是Python里面来定义一个最大数的方法，类似的还可以用 `sys.maxsize` ：

```
class Solution(object):
    def threeSumClosest(self, nums, target):
        """
        :type nums: List[int]
        :rtype: List[List[int]]
        """
        n, res, diff = len(nums), None, float('inf')
        nums.sort() # 先排个序
        for i in range(n):
            if i > 0 and nums[i] == nums[i-1]:
                continue
            l, r = i+1, n-1
            while l < r:
                tmp = nums[i] + nums[l] + nums[r]
                if tmp == target:
                    return target
                elif tmp > target: # 说明当前和大了
                    r -= 1
                    if abs(tmp-target) < diff: # 如果和更接近target了
                        diff = abs(tmp-target)
                        res = tmp
                    while l < r and nums[r] == nums[r+1]:
                        r -= 1
                else: # # 说明当前和小了
                    l += 1
                    if abs(tmp-target) < diff: # 如果和更接近target了
                        diff = abs(tmp-target)
                        res = tmp
                    while l < r and nums[l] == nums[l-1]:
                        l += 1
        return res
```

> `float('inf')` 在 Python 中代表的意思是正无穷， `float('-inf')` 代表负无穷。从数学意义上来说，通常的运算是不会得到 inf 值的。当涉及到 < 和 > 时，所有的数都比 `-inf` 大，所有的数都比 `+inf` 小。

Java 84.18%

```java
class Solution {
    public int threeSumClosest(int[] nums, int target) {
        int n = nums.length;
        // 通过内部 api 进行排序
        Arrays.sort(nums);
        int ret = 1<<30;
        // 通过固定 nums[i]、移动 nums[left] 以及 nums[right] 来逼近目标值 target
        for (int i = 0; i < n; i++) {
            int left = i + 1;
            int right = n - 1;
            while (left < right) {
                int temp = nums[i] + nums[left] + nums[right];
                if (Math.abs(ret - target) > Math.abs(temp - target)) {
                    ret = temp;
                }
                // temp 小于目标值 target 说明应该向由移动 left 指针，如果大于目标值 target 说明应该向左移动 right 指针
                if (temp <= target) {
                    left++;
                } else {
                    right--;
                }
            }
        }
        return ret;
    }
}
```

c++ beats: 50.02%

```cpp
class Solution {
public:
    int threeSumClosest(vector<int>& nums, int target) {
        int n = nums.size();
        sort(nums.begin(), nums.end());
        int ret = 0x3fffffff;
        for (int i = 0;i < n;i++) {
            //对于每个i，通过两个下标来逼近最近target的值
            int left = i + 1;
            int right = n - 1;
            while (left < right) {
                int temp = nums[i] + nums[left] + nums[right];
                if (abs(ret - target) > abs(temp - target)) {
                    ret = temp;
                }
                //如果temp小于target，说明right减下去只会让temp更小，更远离target，此时移动left
                if (temp <= target) {
                    left++;
                } else {
                    right--;
                }
            }
        }
        return ret;
    }
};
```

go beats 44.32%

```go
func abs(x int) int {
    if x < 0 {
        return -x
    } else {
        return x
    }
}

func threeSumClosest(nums []int, target int) int {
    sort.Ints(nums)
    ret := 1 << 30
    n := len(nums)
    for i, v := range nums {
        //对于每个i，通过两个下标来逼近最近target的值
        left := i + 1;
        right := n - 1;
        for left < right {
            temp := v + nums[left] + nums[right]
            if abs(ret - target) > abs(temp - target) {
                ret = temp;
            }
            //如果temp小于target，说明right减下去只会让temp更小，更远离target，此时移动left
            if temp <= target {
                left++
            } else {
                right--
            }
        }
    }
    return ret
}
```

小结

这道题和 3.5 小节其实很像，所以我直接把3.5小节的最优方法拿过来用了一下，算是偷了个懒。其实你在看到很类似的题目的时候也可以像我一样直接把模板拿过来用，有的题目相似到改两个参数就行了，毕竟人生苦短：）

}

19 三数之和

21 电话号码的字母组合