Solve 3-SUM using the *Quadrithmic*, *Quadratic*, and (bonus point) *quadraticWithCalipers* approaches, as shown in skeleton code in the repository. There are hints at the end of Lesson 2.5 Entropy.

There are also hints in the comments of the existing code.  There are a number of unit tests which you should be able to run successfully.

Submit (in your own repository--see instructions elsewhere--include the source code and the unit tests of course):

(a) evidence (screenshot) of your unit tests running (try to show the actual unit test code as well as the green strip);

(b) a spreadsheet showing your timing observations--using the doubling method for at least five values of N--for each of the algorithms (include cubic); Timing should be performed either with an actual stopwatch (e.g. your iPhone) or using the Stopwatch class in the repository.

(c) your brief explanation of why the quadratic method(s) work.

使用 Quadrithmic、Quadratic 和（奖励点）quadraticWithCalipers 方法求解 3-SUM，如存储库中的框架代码所示。 在第 2.5 课熵的末尾有一些提示。

现有代码的注释中也有提示。 您应该能够成功运行许多单元测试。

提交（在您自己的存储库中——参见其他地方的说明——当然包括源代码和单元测试）：

(a) 单元测试运行的证据（屏幕截图）（尝试显示实际的单元测试代码以及绿条）；

(b) 一份电子表格，显示您对每种算法（包括立方）的计时观察结果——对至少五个 N 值使用加倍法； 应使用实际的秒表（例如您的 iPhone）或使用存储库中的 Stopwatch 类来执行计时。

(c) 你对二次方法为何有效的简要解释。

双指针法铺垫： 先将给定 a 排序，复杂度为 O(NlogN)O(NlogN)。

双指针法思路： 固定 33 个指针中最左（最小）数字的指针 k，双指针 i，j 分设在数组索引 (k, len(a))(k,len(a)) 两端，通过双指针交替向中间移动，记录对于每个固定指针 k 的所有满足 a[k] + a[i] + a[j] == 0 的 i,j 组合：

当 a[k] > 0 时直接break跳出：因为 a[j] >= a[i] >= a[k] > 0，即 33 个数字都大于 00 ，在此固定指针 k 之后不可能再找到结果了。

当 k > 0且a[k] == a[k - 1]时即跳过此元素a[k]：因为已经将 a[k - 1] 的所有组合加入到结果中，本次双指针搜索只会得到重复组合。

i，j 分设在数组索引 (k, len(a))(k,len(a)) 两端，当i < j时循环计算s = a[k] + a[i] + a[j]，并按照以下规则执行双指针移动：

当s < 0时，i += 1并跳过所有重复的a[i]；

当s > 0时，j -= 1并跳过所有重复的a[j]；

当s == 0时，记录组合[k, i, j]至res，执行i += 1和j -= 1并跳过所有重复的a[i]和a[j]，防止记录到重复组合。

Double pointer method pavement: first sort the given a, the complexity is O(NlogN)O(NlogN).

The idea of Quadratic method: fix the pointer k of the leftmost (minimum) number among the 3 pointers, and the double pointers i and j are respectively set at both ends of the array index (k, len(a))(k, len(a)).

The pointer moves to the middle alternately, recording all combinations of i,j satisfying a[k] + a[i] + a[j] == 0 for each fixed pointer k:

When a[k] > 0, directly break jumps out: because a[j] >= a[i] >= a[k] > 0, that is, 33 numbers are all greater than 00, and it is impossible to find after this fixed pointer k It worked out.

When k > 0 and a[k] == a[k - 1], this element a[k] is skipped: because all combinations of a[k - 1] have been added to the result, this double pointer search You will only get duplicate combinations.

i, j are set at both ends of the array index (k, len(a))(k, len(a)), and when i < j, loop calculation s = a[k] + a[i] + a[j], And perform a double pointer move according to the following rules:

When s < 0, i += 1 and skip all repeated a[i];

When s > 0, j -= 1 and skip all repetitions of a[j];

When s == 0, record the combination [k, i, j] to res, execute i += 1 and j -= 1 and skip all repeated a[i] and a[j] to prevent recording to repeated combinations .

Author: jyd

Link: https://leetcode.cn/problems/3sum/solution/3sumpai-xu-shuang-zhi-zhen-yi-dong-by-jyd/

Source: LeetCode

Copyright belongs to the author. For commercial reprint, please contact the author for authorization, for non-commercial reprint, please indicate the source.

双指针法铺垫： 先将给定 a 排序

双指针法思路： 固定指针中中间数字的指针j，双指针 i，k分设在j两端，通过双指针交替向中间移动，记录对于每个固定指针 j的所有满足 a[k] + a[i] + a[j] == 0的 j,k组合：

当i < j 且 k > j时循环计算sum = a[k] + a[i] + a[j]，并按照以下规则执行双指针移动：

当sum < 0时，i += 1并跳过所有重复的a[i]；

当sum > 0时，k-= 1并跳过所有重复的a[k]；

当s == 0时，记录组合[i, j, k]至Triples，执行i += 1和k-= 1并跳过所有重复的a[i]和a[j]，防止记录到重复组合。

Pave the way for the double pointer method: first sort the given a

The idea of the double pointer method: The pointer j of the middle number in the fixed pointer, the double pointer i, k are set at both ends of j, and the double pointers are alternately moved to the middle, and record all the conditions that satisfy a[k] + a[ for each fixed pointer j i] + a[j] == 0 j, k combination:

When i < j and k > j, loop to calculate sum = a[k] + a[i] + a[j], and perform double pointer movement according to the following rules:

When sum < 0, i += 1 and skip all repeated a[i];

When sum > 0, k-= 1 and skip all repeated a[k];

When s == 0, record the combination [i, j, k] to Triples, execute i += 1 and k-= 1 and skip all repeated a[i] and a[j] to prevent recording to repeated combinations .

著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。