<https://leetcode.com/problems/count-equal-and-divisible-pairs-in-an-array/description/>

Given a 0-indexed integer array nums of length n and an integer k, return the number of pairs (i, j) where 0 <= i < j < n, such that nums[i] == nums[j] and (i \* j) is divisible by k.

**Example 1:**

**Input:** nums = [3,1,2,2,2,1,3], k = 2

**Output:** 4

**Explanation:**

There are 4 pairs that meet all the requirements:

- nums[0] == nums[6], and 0 \* 6 == 0, which is divisible by 2.

- nums[2] == nums[3], and 2 \* 3 == 6, which is divisible by 2.

- nums[2] == nums[4], and 2 \* 4 == 8, which is divisible by 2.

- nums[3] == nums[4], and 3 \* 4 == 12, which is divisible by 2.

**Example 2:**

**Input:** nums = [1,2,3,4], k = 1

**Output:** 0

**Explanation:** Since no value in nums is repeated, there are no pairs (i,j) that meet all the requirements.

**Constraints:**

1 <= nums.length <= 100

1 <= nums[i], k <= 100

**Attempt 1: 2024-01-17**

**Solution 1: Brute Force (10 min)**

class Solution {

    public int countPairs(int[] nums, int k) {

        int n = nums.length;

        int count = 0;

        for(int i = 0; i < n; i++) {

            for(int j = i + 1; j < n; j++) {

                if(nums[i] == nums[j] && i \* j % k == 0) {

                    count++;

                }

            }

        }

        return count;

    }

}

Time Complexity: O(N^2)

Space Complexity: O(1)

**Solution 2: Greatest Common Dividsor - GCD (180 min)**

class Solution {

    public int countPairs(int[] nums, int k) {

        Map<Integer, List<Integer>> map = new HashMap<>();

        for(int i = 0; i < nums.length; i++) {

            map.putIfAbsent(nums[i], new ArrayList<Integer>());

            map.get(nums[i]).add(i);

        }

        int result = 0;

        for(int num : map.keySet()) {

            Map<Integer, Integer> prev\_GCDs = new HashMap<>();

            for(int index\_a : map.get(num)) {

                int gcd\_a = gcd(index\_a, k);

                for(int gcd\_b : prev\_GCDs.keySet()) {

                    result += (gcd\_a \* gcd\_b % k != 0 ? 0 : prev\_GCDs.get(gcd\_b));

                }

                prev\_GCDs.put(gcd\_a, prev\_GCDs.getOrDefault(gcd\_a, 0) + 1);

            }

        }

        return result;

    }

    private int gcd(int x, int y) {

        if(y == 0) {

            return x;

        }

        return gcd(y, x % y);

    }

}

Refer to

https://leetcode.com/problems/count-equal-and-divisible-pairs-in-an-array/solutions/1784521/o-n-sqrt-n/comments/1285992

Time Complexity: O(N \* tau(N))

Actually, it is O(n \* tau(n)), where tau(n) is the number of divisors of n. Asymptotically, it is much smaller than sqrt(n); square root is sometimes used as a (loose) upper bound for smaller n.

The maximum number of divisors for n <= 100 is 12, and 2304 for n <= 9,999,999,999 (so it becomes more like curt(n)).

Space Complexity: O(N)

**Step by Step**

原理：目标是根据公式If (a \* b) % k == 0, then gcd(a, k) \* gcd(b, k) % k is also 0，

我们需要寻找存在多少组gcd(a, k) \* gcd(b, k) % k == 0

而这里的a和b代表了当nums[a] == nums[b]的时候原数组中nums[a]和nums[b]的下标值分别为a, b

比如 nums = {3,1,2,2,2,1,3}, 那么nums[1] == nums[5] = 1, a = 1, b = 5，所以如果要寻找

一对同时满足nums[a] == nums[b]并且(a \* b) % k == 0的a和b，那必然在同一数值的所有坐标中

寻找，必然 nums = {3,1,2,2,2,1,3} 构建的如下Map中，我们需要在数值1的情况下在{1,5}中寻找

a和b，同理，在数值2的情况下在{2,3,4}中寻找a和b，在数值3的情况下在{0,6}中寻找a和b

gcds是每一轮中保存当前轮所有key == nums[a]时获得的所有gcdi (等价于公式中的gcd(a, k))的frequency

Map

1 -> {1, 5}

2 -> {2, 3, 4}

3 -> {0, 6}

=====================================================================================

首先当key == nums[1] = 1时，我们获得所有下标{1,5}，然后在{1,5}中寻找a和b：

对1做gcd(1, 2)获得gcdi = 1，此时因为gcds中没有任何之前的gcdi存入，所以为empty map {{}},

所以并没有gcdj (等价于公式中的gcd(b, k)), 因此result并不增加依然为0，随后我们需要在gcds中

更新gcdi的frequency，也就是将当前处理gcd(1, 2)所获得的gcdi = 1的频率更新到gcds中，在后续

的遍历中将作为匹配其他gcdi的gcdj使用，gcds = {{1,1}}

对5做gcd(5, 2)获得gcdi = 1，此时因为gcds中已经有了之前处理gcd(1, 2)时获得的gcdi = 1存入的

{{1,1}}，所以gcdj = 1，但以为gcdi \* gcdj % k != 0，所以result并不增加依然为0，随后我们需

要在gcds中更新gcdi的frequency，也就是将当前处理gcd(5, 2)所获得的gcdi = 1的频率更新到gcds

中，在后续的遍历中将作为匹配其他gcdi的gcdj使用，gcds = {{1,2}}

-------------------------------------------------------------------------------------

key = 1 -> map gcds = {{}}

map.get(1) = {1, 5}

it = 1

gcdi = gcd(it, k) = gcd(1, 2) = 1

gcds.keySet() -> empty {}

gcdj not exist

gcds.put(gcdi, gcds.getOrDefault(gcdi, 0) + 1) = {{1, 1}}

it = 5

gcdi = gcd(it, k) = gcd(5, 2) = 1

gcds.keySet() -> {1}

gcdj = 1

res += (gcdi \* gcdj % k != 0) ? 0 : gcds.get(gcdj) -> res += (1 \* 1 % 2 != 0) ? 0 : gcds.get(1) -> res += 0 = 0

gcds.put(gcdi, gcds.getOrDefault(gcdi, 0) + 1) = {{1, 2}}

=====================================================================================

然后当key == nums[2] = 2时，我们获得所有下标{2,3,4}，然后在{2,3,4}中寻找a和b：

对2做gcd(2, 2)获得gcdi = 2，此时因为gcds中没有任何之前的gcdi存入，所以为empty map {{}},

所以并没有gcdj (等价于公式中的gcd(b, k)), 因此result并不增加依然为0，随后我们需要在gcds中

更新gcdi的frequency，也就是将当前处理gcd(2, 2)所获得的gcdi = 2的频率更新到gcds中，在后续

的遍历中将作为匹配其他gcdi的gcdj使用，gcds = {{2,1}}

对3做gcd(3, 2)获得gcdi = 1，此时因为gcds中已经有了之前处理gcd(2, 2)时获得的gcdi = 2存入的

{{2,1}}，所以gcdj = 2，以为gcdi \* gcdj % k == 0，所以result增加为0 + gcds.get(j)也即

0 + gcds.get(2) = 1，随后我们需要在gcds中更新gcdi的frequency，也就是将当前处理gcd(5, 2)

所获得的gcdi = 1的频率更新到gcds中，在后续的遍历中将作为匹配其他gcdi的gcdj使用，gcds =

{{1,1},{2,1}}

对4做gcd(4, 2)获得gcdi = 2，此时因为gcds中已经有了之前处理gcd(2, 2)和gcd(3, 2)时获得的

gcdi = 2存入的{{2,1}}和gcdi = 1存入的{{1,1}}，所以gcdj = 2和1，因为gcdi \* gcdj % k == 0，

所以result增加为1 + gcds.get(j)也即1 + gcds.get(2) = 2，2 + gcds.get(j)也即2 + gcds.get(1) = 3，

随后我们需要在gcds中更新gcdi的frequency，也就是将当前处理gcd(4, 2)所获得的gcdi = 2的频率

更新到gcds中，在后续的遍历中将作为匹配其他gcdi的gcdj使用，gcds = {{1,1},{2,2}}

... etc.

-------------------------------------------------------------------------------------

key = 2 -> map gcds = {{}}

map.get(2) = {2, 3, 4}

it = 2

gcdi = gcd(it, k) = gcd(2, 2) = 2

gcds.keySet() -> empty {}

gcdj not exist

gcds.put(gcdi, gcds.getOrDefault(gcdi, 0) + 1) = {{2, 1}}

it = 3

gcdi = gcd(it, k) = gcd(3, 2) = 1

gcds.keySet() -> {2}

gcdj = 2

res += (gcdi \* gcdj % k != 0) ? 0 : gcds.get(gcdj) -> res += (1 \* 2 % 2 != 0) ? 0 : gcds.get(2) -> res += 1 = 1

gcds.put(gcdi, gcds.getOrDefault(gcdi, 0) + 1) = {{1, 1}, {2, 1}}

it = 4

gcdi = gcd(it, k) = gcd(4, 2) = 2

gcds.keySet() -> {1, 2}

gcdj = 1

res += (gcdi \* gcdj % k != 0) ? 0 : gcds.get(gcdj) -> res += (2 \* 1 % 2 != 0) ? 0 : gcds.get(1) -> res += 1 = 2

gcdj = 2

res += (gcdi \* gcdj % k != 0) ? 0 : gcds.get(gcdj) -> res += (2 \* 2 % 2 != 0) ? 0 : gcds.get(2) -> res += 1 = 3

gcds.put(gcdi, gcds.getOrDefault(gcdi, 0) + 1) = {{1, 1}, {2, 2}}

... etc.

=====================================================================================

**Refer to**

<https://leetcode.com/problems/count-equal-and-divisible-pairs-in-an-array/solutions/1784521/o-n-sqrt-n/>

Can we do better than brute-force? We will need it to solve a follow up problem (check this post for intuition):

**L2183.Count Array Pairs Divisible by K**

**=> If (a \* b) % k == 0, then gcd(a, k) \* gcd(b, k) % k is also 0**

For each number, collect indexes in the increasing order.

For a number, go through its indexes:

Track count of each gcd(j, k) we encountered so far in a map.

For each index i, check its gcd(i, k) against GCDs for previous indices.

Add count to the result if gcd(i, k) \* gcd(j, k) % k == 0.

**Java version (convert by chatGPT)**

class Solution {

    public int countPairs(int[] nums, int k) {

        Map<Integer, ArrayList<Integer>> map = new HashMap<>();

        int res = 0;

        for (int i = 0; i < nums.length; i++) {

            if (map.containsKey(nums[i])) {

                map.get(nums[i]).add(i);

            } else {

                map.put(nums[i], new ArrayList<Integer>());

                map.get(nums[i]).add(i);

            }

        }

        for (Integer key : map.keySet()) {

            Map<Integer, Integer> gcds = new HashMap<>();

            for (Integer it : map.get(key)) {

                int gcdi = gcd(it, k);

                for (Integer gcdj : gcds.keySet()) {

                    res += (gcdi \* gcdj % k != 0) ? 0 : gcds.get(gcdj);

                }

                gcds.put(gcdi, gcds.getOrDefault(gcdi, 0) + 1);

            }

        }

        return res;

    }

    private static int gcd(int a, int b) {

        if (b == 0)

            return a;

        return gcd(b, a % b);

    }

}

**Original C++ version**

int countPairs(vector<int>& nums, int k) {

int res = 0;

unordered\_map<int, vector<int>> m;

for (int i = 0; i < nums.size(); ++i)

m[nums[i]].push\_back(i);

for (auto &[n, ids] : m) {

unordered\_map<int, int> gcds;

for (auto i : ids) {

auto gcd\_i = gcd(i, k);

for (auto &[gcd\_j, cnt] : gcds)

res += gcd\_i \* gcd\_j % k ? 0 : cnt;

++gcds[gcd\_i];

}

}

return res;

}