<https://leetcode.com/problems/3sum-with-multiplicity/description/>

Given an integer array arr, and an integer target, return the number of tuples i, j, k such that i < j < k and arr[i] + arr[j] + arr[k] == target.

As the answer can be very large, return it modulo 10^9 + 7.

**Example 1:**

**Input:** arr = [1,1,2,2,3,3,4,4,5,5], target = 8

**Output:** 20

**Explanation:** Enumerating by the values (arr[i], arr[j], arr[k]):(1, 2, 5) occurs 8 times;(1, 3, 4) occurs 8 times;(2, 2, 4) occurs 2 times;(2, 3, 3) occurs 2 times.

**Example 2:**

**Input:** arr = [1,1,2,2,2,2], target = 5

**Output:** 12

**Explanation:** arr[i] = 1, arr[j] = arr[k] = 2 occurs 12 times:We choose one 1 from [1,1] in 2 ways,and two 2s from [2,2,2,2] in 6 ways.

**Example 3:**

**Input:** arr = [2,1,3], target = 6

**Output:** 1

**Explanation:** (1, 2, 3) occured one time in the array so we return 1.

**Constraints:**

3 <= arr.length <= 3000

0 <= arr[i] <= 100

0 <= target <= 300

**Attempt 1: 2024-06-13**

**Solution 1: Three Pointers (60 min)**

class Solution {

    public int threeSumMulti(int[] A, int target) {

        Arrays.sort(A);

        int mod = 1000000007;

        int result = 0;

        for(int i = 0; i < A.length - 2; i++) {

            int lo = i + 1;

            int hi = A.length - 1;

            long count = 0;

            while(lo < hi) {

                if(A[lo] + A[hi] == target - A[i]) {

                    // Count repeat elements from two directions, and

                    // their production means how many tuples result same

                    if(A[lo] != A[hi]) {

                        long countL = 1;

                        long countR = 1;

                        while(lo + 1 < hi && A[lo + 1] == A[lo]) {

                            countL++;

                            lo++;

                        }

                        while(hi - 1 > lo && A[hi - 1] == A[hi]) {

                            countR++;

                            hi--;

                        }

                        count += (countL \* countR) % mod;

                        lo++;

                        hi--;

                    } else {

                        // Since we already sort the array if A[lo] == A[hi] means

                        // all elements between index lo and hi in A are same value

                        long n = hi - lo + 1;

                        count += (n \* (n - 1) / 2) % mod;

                        // No need to continue current loop since all elements are same

                        break;

                    }

                } else if(A[lo] + A[hi] > target - A[i]) {

                    hi--;

                } else {

                    lo++;

                }

            }

            result = (int)(result + count) % mod;

        }

        return result;

    }

}

Time complexity: O(N^2)

Space complexity: O(1)

**Refer to**

<https://leetcode.com/problems/3sum-with-multiplicity/discuss/181080/Java-sorting-solution-O(N2)>

The solution is similar to 3Sum problem (<https://leetcode.com/submissions/detail/128547721/).>

The key is how to skip duplicate numbers:

Case 1: A[l] == A[r], the total possible answer is num \* (num-1) / 2, where num = r-l+1.

Case 2: A[l] != A[r], find how many duplicates in the left-hand side (which is cntL) and how many duplicates in the right-hand side (which is cntR). Then, search for the next possible answers.

class Solution {

public int threeSumMulti(int[] A, int target) {

int mod = 1000000007;

int ans = 0;

Arrays.sort(A);

for(int i = 0; i < A.length-2; i++) {

long cnt = 0;

int l = i+1, r = A.length-1;

while(l < r) {

if(A[i] + A[l] + A[r] > target) r--;

else if(A[i] + A[l] + A[r] < target) l++;

else {

if(A[l] != A[r]) {

long cntL = 1, cntR = 1;

while(l+1 < r && A[l] == A[l+1]) { cntL++; l++; }

while(l < r-1 && A[r] == A[r-1]) { cntR++; r--; }

cnt += (cntL \* cntR) % mod;

l++;

r--;

} else {

long n = r-l+1;

cnt += (n \* (n-1) / 2) % mod;

break;

}

}

}

ans = (int) (ans + cnt) % mod;

}

return ans;

}

}

Time complexity: O(N^2)

Space complexity: O(1)

**Solution 2: Two Pointers + Hash Table (60 min)**

class Solution {

    public int threeSumMulti(int[] A, int target) {

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

        int result = 0;

        int mod = 1000000007;

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

            result = (result + map.getOrDefault(target - A[i], 0)) % mod;

            for(int j = 0; j < i; j++) {

                map.put(A[j] + A[i], map.getOrDefault(A[j] + A[i], 0) + 1);

            }

        }

        return result;

    }

}

Time complexity: O(N^2)

Space complexity: O(1)

**Refer to**

<https://leetcode.com/problems/3sum-with-multiplicity/solutions/181128/10-lines-Super-Super-Easy-Java-Solution/>

**Think Outside of The Box!**

Intuitively, you will try to solve it based on the solution of 3Sum.

But... Build a map for counting different sums of two numbers. The rest of things are straightfoward.

class Solution {

public int threeSumMulti(int[] A, int target) {

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

int res = 0;

int mod = 1000000007;

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

res = (res + map.getOrDefault(target - A[i], 0)) % mod;

for (int j = 0; j < i; j++) {

int temp = A[i] + A[j];

map.put(temp, map.getOrDefault(temp, 0) + 1);

}

}

return res;

}

}

**Solution 3: Three Pointers + Hash Table (60 min)**

class Solution {

    public int threeSumMulti(int[] A, int target) {

        long[] c = new long[101];

        for (int a : A) c[a]++;

        long res = 0;

        for (int i = 0; i <= 100; i++)

            for (int j = i; j <= 100; j++) {

                int k = target - i - j;

                if (k > 100 || k < 0) continue;

                if (i == j && j == k)

                    res += c[i] \* (c[i] - 1) \* (c[i] - 2) / 6;

                else if (i == j && j != k)

                    res += c[i] \* (c[i] - 1) / 2 \* c[k];

                else if (j < k)

                    res += c[i] \* c[j] \* c[k];

            }

        return (int)(res % (1e9 + 7));

    }

}

Time Complexity:

3 <= A.length <= 3000, so N = 3000

But 0 <= A[i] <= 100

So my solution is O(N + 101 \* 101)

**Refer to**

<https://leetcode.com/problems/3sum-with-multiplicity/solutions/181131/c-java-python-o-n-101-101/>

Count the occurrence of each number.

using hashmap or array up to you.

Loop i on all numbers,

loop j on all numbers,

check if k = target - i - j is valid.

Add the number of this combination to result.

3 cases covers all possible combination:

i == j == k

i == j != k

i < k && j < k

Time Complexity:

3 <= A.length <= 3000, so N = 3000

But 0 <= A[i] <= 100

So my solution is O(N + 101 \* 101)

<https://leetcode.com/problems/3sum-with-multiplicity/solutions/181131/c-java-python-o-n-101-101/comments/481912>

Tried to explain the three cases. The task is to pick 3 numbers, now you do not want to name it i, j, k, just three numbers in mind.

Case 1: three numbers are the same. Then you will have i == j == k.

Case 2: two of the three numbers are equal. For example, 5, 4, 5. To avoid duplicates, you can choose to make i=5, j=5, k = 4. Once this is chosen, you do not want to have others (e.g. i = 5, k =5, j = 4, or j = 5, k =5, i =4). Otherwise, You will make duplicates. In brief, because two numbers are the same, you only want to have either i, j, or j, k, or k, i to be the same numbers. In lee215's solution, he picked up i, j.

public int threeSumMulti(int[] A, int target) {

long[] c = new long[101];

for (int a : A) c[a]++;

long res = 0;

for (int i = 0; i <= 100; i++)

for (int j = i; j <= 100; j++) {

int k = target - i - j;

if (k > 100 || k < 0) continue;

if (i == j && j == k)

res += c[i] \* (c[i] - 1) \* (c[i] - 2) / 6;

else if (i == j && j != k)

res += c[i] \* (c[i] - 1) / 2 \* c[k];

else if (j < k)

res += c[i] \* c[j] \* c[k];

}

return (int)(res % (1e9 + 7));

}

**Refer to**

[L15.P3.4.3Sum (Ref.L18)](note://EC0082E510FD4360AD113BFC40B23D1F)