Three Sum Problem

by jitsceait / May 26, 2020

Given an array *nums* of *n* integers, are there elements *a*, *b*,*c* in *nums* such that *a*+ *b* + *c*= 0? Find all unique triplets in the array which gives the sum of zero.

The solution set must not contain duplicate triplets.

This problem is commonly known as three sum problem.

```
Input: = [-1, 0, 1, 2, -1, -4],
Output:
[
  [-1, 0, 1],
  [-1, -1, 2]
]
```

Three Sum problem : thought process

Before going into the details of find three numbers with a given sum, do you know how to find two numbers with a given sum *s*? The idea is simple, we keep a map of all the numbers in we already seen, if see a number *a[i]*, we see in the map if *S-a[i]* is present or not. If yes, then we found the pair, if not, we put *a[i]* and move forward.

This solution has linear time complexity i.e *O(n)* with a space complexity of *O(n)* as well. There is a way to avoid space complexity by sorting the input array first and using two pointer approach.

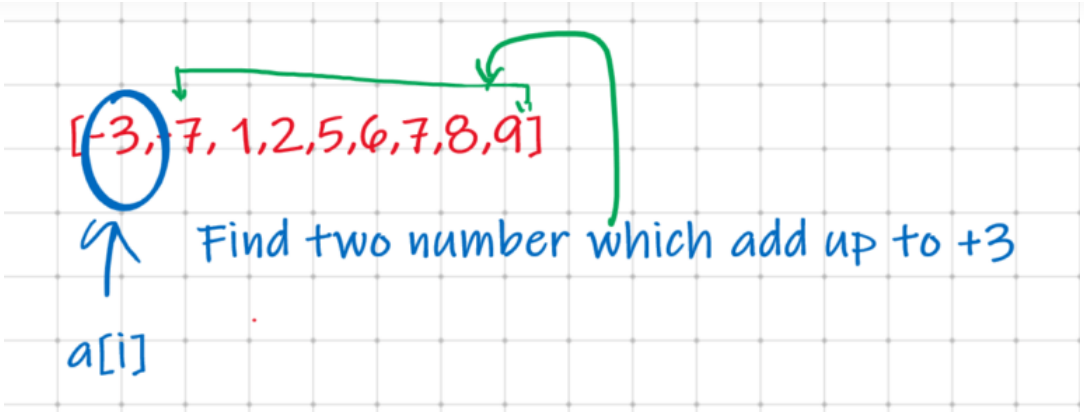
Details of 2 sum problem, you can read here: 2 Sum problem: Find pair with given sum in array

Can we use our understanding of the two-sum problem to solve this three-sum problem?

Hint 1:
What happens if we take one number from the array and say that this number will be one of the three numbers which add up to zero? What happens to our problem statement now?

Hint 2:
If you have decided that *A[0]* will be part of three numbers which add up to zero, all we need is two find out two numbers from index *1..len-1* which add up to *O – A[0]*. What does that mean? It looks like a 2-sum problem, doesn't it?

[expand title="**Can you please explain more?**"]
We will start with index *i* from 0 and len-1, each time claiming that *A[i]* is part of the solution. Once, we fixed *A[i]* in solution, we will find



[/expand]

[expand title="Show me the implementation of three sum problem" tag="h3"]

```
class Solution {

        public List<List<Integer>> threeSum(int[] nums) {

            List<List<Integer>> result = new ArrayList<>();
            //O(n log n)
            Arrays.sort(nums);


            for(int i=0; i<nums.length; i++){ if (i > 0 && nums[i] == nums[i - 1]) {
                 continue;
                }
                if(nums[i] > 0) continue;
                int j = i+1;
                int k = nums .length -1;

```

```
                while(j<k){
                    if(nums[j] + nums[k] == target ){
                      /* If find j and k, such that element at
                         index i, j and K add to zero, put them in result
                      */
                        ArrayList<Integer> triplet = new ArrayList<Integer>(
                            Arrays.asList(nums[i], nums[j], nums[k]));
                        j++;
                        k--;
                         while (j < k && nums[j] == nums[j - 1]) j++;  // skip same result
                         while (j < k && nums[k] == nums[k + 1]) k--; // skip same result //since we wa
                        k--;
                    }
                    else{
                        j++;
                    }

                }
            }
            return  result;
        }
    }
```

[/expand]

If you do not come up with the idea of moving *i*, *j* and *k* to get unique triplets, it is OK. You can always use HashSet to filter out duplicates and then copy the HashSet to the result list. It will take more time but will give the correct result.

```
if(uniqueSet.add(triplet)){
    result.add(triplet);
}
```

Can you solve the following problems on the leetcode?

1. 3SUM
2. 3Sum Closest
3. Four sum problem.

Tags: 2 SUM PROBLEM 3 SUM PROBLEM ARRAYS

Search Website

Search for...

System design basics-IV: Queues

Sort a K Sorted Array

Facebook interview questions

Google interview questions

Privacy policy

Maximum Path Sum in Binary Tree

Spiral traversal of a matrix

Neve | Powered by WordPress