LearnersBucket

3 sum problem algorithm

Posted on September 13, 2020 | by Prashant Yadav

Posted in Algorithms, Arrays | Tagged medium

Given an array of integers and a target we have to check if there exists a triplet such that their combined sum is equal to the given target. Popularly known as 3 sum problem algorithm.

Example

```
Input:
arr = [1, 2, 3, 5, 6, 11, 15, 16, 17, 18];
sum = 20;

Output:
true
[1, 16, 3]
```


This is the second question from the series

- 2 sum
- 3 sum
- 4 sum

Naïve brute force solution

3 nested loops can be used to consider every triplets in the given array and determine if desired sum is found.

Naïve recursive solution

Using recursion to solve this.

- For each item we either consider the current item or ignore it and then recur for the remaining items.
- If triplet with given target is found return true, otherwise return false.

```
const threeSum = (arr, n, sum, count) => {
    //if there exits a triplet with given sum
    if(sum === 0 && count === 3){
      return true;
    }

    //if there is no triplet with desired sum
    if(count === 3 || n === 0 || sum < 0){
      return false;
    }

    //keep recur by
    //considering current element or
    //ignoring current element
    return threeSum(arr, n - 1, sum - arr[n - 1], count + 1) || threeSum(arr, n - 1, sum, count)
}
```

Copy

```
Input:
const arr = [1, 2, 3, 5, 6, 11, 15, 16, 17, 18];
const sum = 20;
console.log(threeSum(arr, arr.length, sum, 0));

Output:
true
```

Copy

Time complexity: (2 ^ N) because for each function call we are calling the same function twice.
Space complexity: (2 ^ N) considering the call stack, O(1) other wise.

Efficient approach using hashing to solve 3 sum problem algorithm.

In this approach we assume that the triplet can be broken down as a pair plus one extra element. Thus we can use the 2 sum algorithm's logic to solve it.

Conceptually this is how it works.

- We store each element from the array in the map along with its index.
- Then we consider all the pair in the array and check if the map has the remaining sum or not.

- If the index of the remaining sum is not overlapping with the pair indexes then triplets exists thus return true, false otherwise.

```
const threeSumWithHashing = (arr, sum) => {
  const map = new Map();

  // Add element along with its index in the hashmap
  for(let i = 0; i < arr.length; i++){
    map.set(arr[i], i);
  }

  // check each pair
  // we ignore last element because we need a triplet (pair + 1 element).
  for(let i = 0; i < arr.length - 1; i++){

    for(let j = i + 1; j < arr.length; j++){

       //pending sum
       let val = sum - (arr[i] + arr[j]);

       // there is a triplet, if we pending sum is available in the map
       if(map.has(val)){
          // if it is unique, return true
          if (map.get(val) !== i && map.get(val) !== j) {
             return true;
          }
        }
      }
    }

    // return false, if there is no triplet
    return false;
}
```

```
Input:
const arr = [1, 2, 3, 5, 6, 11, 15, 16, 17, 18];
const sum = 20;
console.log(threeSumWithHashing(arr, sum));

Output:
true
```

Time complexity: O(n + n ^ 2) = O(n ^ 2).

Space complexity: O(n).

Printing all the unique triplets that are possible in 3 sum problem algorithm

To print all the unique triplets we use the same logic as we have used to check if triplet exists or not with little improvements.

First, sort the array in ascending order so that we don't repeat the combination and then for each element in the array we check if triplets exists by adding the current element and pair from the subarray.

```
const printAllTriplets = (arr, sum) => {
  // increasing order
  arr = arr.sort((a, b) => a - b);

  // explore all triplets
  for(let i = 0; i <= arr.length - 3; i++){

    // calc pending sum
    let k = sum - arr[i];

    // with two pointers, explore all the pairs of subarray for triplets
    let low = i + 1;
    let high = arr.length - 1;

    while (low < high){

        // if total < pending sum, increment lower index
        if (arr[low] + arr[high] < k) {
               low++;
        }

        // if total > pending sum, decrement higher index
        else if (arr[low] + arr[high] > k) {
               high--;
        }

        // otherwise triplet is found
        else {
               //print
               console.log(arr[i], arr[low], arr[high]);

               // change the indexes
               low++;
               high--;
        }
     }
  }
}
```

```
Input:
const arr = [1, 2, 3, 5, 6, 11, 15, 16, 17, 18];
const sum = 20;
printAllTriplets(arr, sum);

Output:
1 16 3
1 17 2
```

Time complexity: O(n ^ 2).

Space complexity: O(1).

Recommended Posts:

Find numbers that appear twice in an array.

Learn how to reverse a linked list

Find longest palindrome in a string

Find the maximum depth of nested parentheses in a string

Swap two numbers without temp variables

Longest Common Subsequence

Difference between square of sum of numbers and sum of square of numbers.

Selection sort in javascript

Learn how to reverse words in a string

Program to sort only positive numbers of the array

Prev Next