Find All Four Sum Numbers

Thursday, December 23, 2021 12:37 PM

Find All Four Sum Numbers

Medium Accuracy: 41.1% Submissions: 42220 Points: 4

Given an array of integers and another number. Find all the **unique** quadruple from the given array that sums up to the given number.

Example 1:

Input:
N = 5, K = 3
A[] = {0,0,2,1,1}
Output: 0 0 1 2 $
Explanation: Sum of 0, 0, 1, 2 is equal to K.

Example 2:

Input:
N = 7, K = 23
A[] = {10,2,3,4,5,7,8}
Output: 2 3 8 10 $2 4 7 10 $3 5 7 8 $
Explanation: Sum of 2, 3, 8, 10 = 23, sum of 2, 4, 7, 10 = 23 and sum of 3, 5, 7, 8 = 23.

Your Task:

You don't need to read input or print anything. Your task is to complete the function **fourSum()** which takes the array arr[] and the integer k as its input and returns an array containing all the quadruples in a lexicographical manner. Also note that all the quadruples should be internally sorted, ie for any quadruple [q1, q2, q3, q4] the following should follow: q1 <= q2 <= q3 <= q4. (In the output each quadruple is separate by $. The printing is done by the driver's code)

From <https://practice.geeksforgeeks.org/problems/find-all-four-sum-numbers1732/1>

Expected Time Complexity: $O(N^3)$.
Expected Auxiliary Space: $O(N^2)$.

Constraints:
1 <= N <= 100
-1000 <= K <= 1000
-100 <= A[] <= 100

View Bookmarked Problems

Company Tags

Adobe Amazon Google Microsoft OYO Rooms

Topic Tags

Related Courses

Amazon SDE Test Series Google Test Series Placement Guide 2021-2022

SOLUTION

```
class Solution {
   public ArrayList<ArrayList<Integer>> fourSum(int[] a, int k) {
      int n = a.length;
      ArrayList<ArrayList<Integer>> ans = new ArrayList<ArrayList<Integer>>();
      if (n < 4) return ans;

Arrays.sort(a);
      for (int i = 0; i < n - 3; i++) {
         // current element is greater than k then no quadruplet can be found
         if (a[i] > 0 && a[i] > k) break;

// removing duplicates
         if (i > 0 && a[i] == a[i - 1]) continue;

for (int j = i + 1; j < n - 2; ++j) {
            // removing duplicates
            if (j > i + 1 && a[j] == a[j - 1]) continue;

// taking two pointers
            int left = j + 1;
            int right = n - 1;
            while (left < right) {
               int old_l = left;
               int old_r = right;
               // calculate current sum
               int sum = a[i] + a[j] + a[left] + a[right];
               if (sum == k) {
                  // add to answer
                  ans.add(new ArrayList<Integer>(
```

```
                    Arrays.asList(a[i], a[j], a[left], a[right])));

// removing duplicates
                while (left < right && a[left] == a[old_l]) left++;
                while (left < right && a[right] == a[old_r]) right--;
              } else if (sum > k) {
                right--;
              } else {
                left++;
              }
            }
          }
        }
      return ans;
    }
}
```

From <https://practice.geeksforgeeks.org/problems/find-all-four-sum-numbers1732/1#>

Find four elements that sum to a given value | Set 2

- Difficulty Level : Hard
- Last Updated : 24 Sep, 2021

Given an array of integers, find anyone combination of four elements in the array whose sum is equal to a given value X.

For example,


```
Input: array = {10, 2, 3, 4, 5, 9, 7, 8}
       X = 23
Output: 3 5 7 8
Sum of output is equal to 23,
i.e. 3 + 5 + 7 + 8 = 23.
Input: array = {1, 2, 3, 4, 5, 9, 7, 8}
       X = 16
Output: 1 3 5 7
Sum of output is equal to 16,
i.e. 1 + 3 + 5 + 7 = 16.
```

Recommended: Please solve it on "***PRACTICE***" first, before moving on to the solution.

We have discussed an **O(n^3)** algorithm in the previous post on this topic. The problem can be solved in **O(n^2 Logn)** time with the help of auxiliary space.

Thanks to itsnimish for suggesting this method. Following is the detailed process.

Method 1: Two Pointers Algorithm.

Approach: Let the input array be A[].

1. Create an auxiliary array aux[] and store sum of all possible pairs in aux[]. The size of aux[] will be n*(n-1)/2 where n is the size of A[].
2. Sort the auxiliary array aux[].

3. Now the problem reduces to find two elements in aux[] with sum equal to X. We can use method 1 of this post to find the two elements efficiently. There is following important point to note though:
   An element of aux[] represents a pair from A[]. While picking two elements from aux[], we must check whether the two elements have an element of A[] in common. For example, if first element sum of A[1] and A[2], and second element is sum of A[2] and A[4], then these two elements of aux[] don't represent four distinct elements of input array A[].
   Below is the implementation of the above approach:

- C++
- C

```
// C++ program to find 4 elements
// with given sum
#include <bits/stdc++.h>
using namespace std;

// The following structure is needed
// to store pair sums in aux[]
class pairSum {
public:
    // index (int A[]) of first element in pair
    int first;

    // index of second element in pair
    int sec;

    // sum of the pair
    int sum;
};

// Following function is needed
// for library function qsort()
int compare(const void* a, const void* b)
{
    return ((*(pairSum*)a).sum - (*(pairSum*)b).sum);
}

// Function to check if two given pairs
// have any common element or not
bool noCommon(pairSum a, pairSum b)
{
    if (a.first == b.first || a.first == b.sec
        || a.sec == b.first || a.sec == b.sec)
        return false;
    return true;
}

// The function finds four
// elements with given sum X
void findFourElements(int arr[], int n, int X)
{
    int i, j;

    // Create an auxiliary array
    // to store all pair sums
    int size = (n * (n - 1)) / 2;
    pairSum aux[size];

    // Generate all possible pairs
    // from A[] and store sums
    // of all possible pairs in aux[]
    int k = 0;
    for (i = 0; i < n - 1; i++) {
```

```
            for (j = i + 1; j < n; j++) {
                aux[k].sum = arr[i] + arr[j];
                aux[k].first = i;
                aux[k].sec = j;
                k++;
            }
        }

        // Sort the aux[] array using
        // library function for sorting
        qsort(aux, size, sizeof(aux[0]), compare);

        // Now start two index variables
        // from two corners of array
        // and move them toward each other.
        i = 0;
        j = size - 1;
        while (i < size && j >= 0) {
            if ((aux[i].sum + aux[j].sum == X)
                && noCommon(aux[i], aux[j])) {
                cout << arr[aux[i].first] << ", "
                     << arr[aux[i].sec] << ", "
                     << arr[aux[j].first] << ", "
                     << arr[aux[j].sec] << endl;
                return;
            }
            else if (aux[i].sum + aux[j].sum < X)
                i++;
            else
                j--;
        }
    }

    // Driver code
    int main()
    {
        int arr[] = { 10, 20, 30, 40, 1, 2 };
        int n = sizeof(arr) / sizeof(arr[0]);
        int X = 91;

        // Function Call
        findFourElements(arr, n, X);
        return 0;
    }

    // This is code is contributed by rathbhupendra
```

Output

```
20, 1, 30, 40
```

Please note that the above code prints only one quadruple. If we remove the return statement and add statements “i++; j–;”, then it prints same quadruple five times. The code can modified to print all quadruples only once. It has been kept this way to keep it simple.

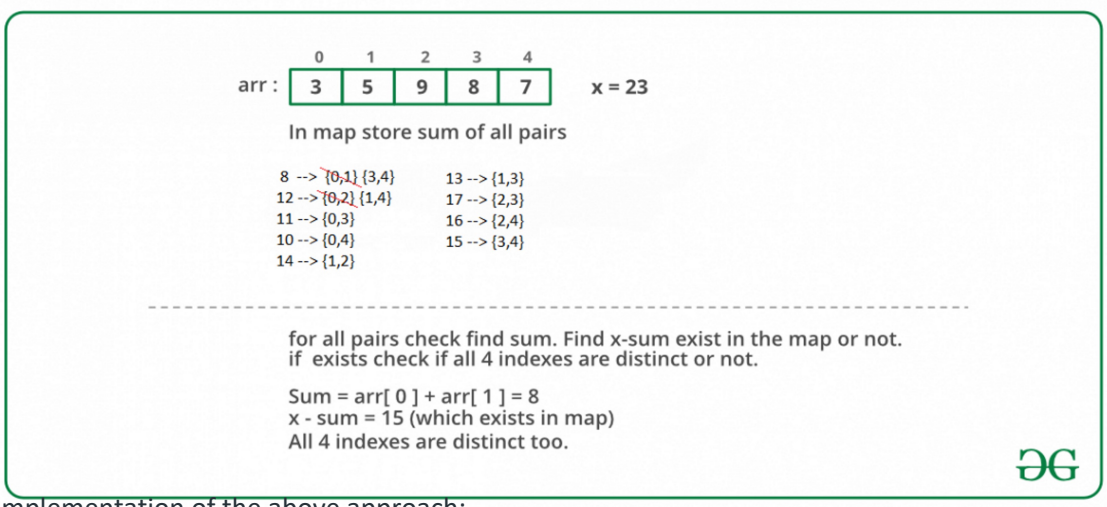
Complexity Analysis:

- **Time complexity:** O(n^2Logn).
  The step 1 takes O(n^2) time. The second step is sorting an array of size O(n^2). Sorting can be done in O(n^2Logn) time using merge sort or heap sort or any other O(nLogn) algorithm. The third step takes O(n^2) time. So overall complexity is O(n^2Logn).
- **Auxiliary Space:** O(n^2).
  The size of the auxiliary array is O(n^2). The big size of the auxiliary array can be a concern in this method.

Method 2: Hashing Based Solution[$O(n^2)$]

Approach:

1. Store sums of all pairs in a hash table
2. Traverse through all pairs again and search for **X – (current pair sum)** in the hash table.
3. If a pair is found with the required sum, then make sure that all elements are distinct array elements and an element is not considered more than once.
   Below image is a dry run of the above approach:



Below is the implementation of the above approach:

- C++
- Java
- Python3
- C#
- Javascript

Below is the implementation of the above approach:

- C++
- Java
- Python3
- C#
- Javascript

From <https://www.geeksforgeeks.org/find-four-elements-that-sum-to-a-given-value-set-2/?ref=lbp>

```
// A hashing based Java program to find
// if there are four elements with given sum.
import java.util.HashMap;
class GFG {
    static class pair {
        int first, second;
        public pair(int first, int second)
        {
            this.first = first;
            this.second = second;
        }
    }

    // The function finds four elements
    // with given sum X
    static void findFourElements(int arr[], int n, int X)
    {
        // Store sums of all pairs in a hash table
        HashMap<Integer, pair> mp
            = new HashMap<Integer, pair>();
        for (int i = 0; i < n - 1; i++)
            for (int j = i + 1; j < n; j++)
                mp.put(arr[i] + arr[j], new pair(i, j));

        // Traverse through all pairs and search
        // for X - (current pair sum).
        for (int i = 0; i < n - 1; i++) {
            for (int j = i + 1; j < n; j++) {
                int sum = arr[i] + arr[j];

                // If X - sum is present in hash table,
```

```
                if (mp.containsKey(X - sum)) {

                    // Making sure that all elements are
                    // distinct array elements and an
                    // element is not considered more than
                    // once.
                    pair p = mp.get(X - sum);
                    if (p.first != i && p.first != j
                        && p.second != i && p.second != j) {
                        System.out.print(
                            arr[i] + ", " + arr[j] + ", "
                            + arr[p.first] + ", "
                            + arr[p.second]);
                        return;
                    }
                }
            }
        }
    }

    // Driver Code
    public static void main(String[] args)
    {
        int arr[] = { 10, 20, 30, 40, 1, 2 };
        int n = arr.length;
        int X = 91;

        // Function call
        findFourElements(arr, n, X);
    }
}

// This code is contributed by Princi Singh
```

Output

```
20, 30, 40, 1
```

Complexity Analysis:

- **Time complexity:** O(n^2).
  Nested traversal is needed to store all pairs in the hash Map.
- **Auxiliary Space:** O(n^2).
  All n*(n-1) pairs are stored in hash Map so the space required is O(n^2)
  Please write comments if you find any of the above codes/algorithms incorrect, or find other ways to solve the same problem.
  Method 3: Solution having no duplicate elements

  Approach:

1. Store sums of all pairs in a hash table
2. Traverse through all pairs again and search for X – (current pair sum) in the hash table.
3. Consider a temp array that is initially stored with zeroes. It is changed to 1 when we get 4 elements that sum up to the required value.
4. If a pair is found with the required sum, then make sure that all elements are distinct array elements and check if the value in temp array is 0 so that duplicates are not considered.

From <https://www.geeksforgeeks.org/find-four-elements-that-sum-to-a-given-value-set-2/?ref=lbp>

Below is the implementation of the code:

- C++
- Java
- Python3
- C#
- Javascript

```java
// Java program to find four
// elements with the given sum
import java.util.*;

class fourElementWithSum {

    // Function to find 4 elements that add up to
    // given sum
    public static void fourSum(int X, int[] arr,
                               Map<Integer, pair> map)
    {
        int[] temp = new int[arr.length];

        // Iterate from 0 to temp.length
        for (int i = 0; i < temp.length; i++)
            temp[i] = 0;

        // Iterate from 0 to arr.length
        for (int i = 0; i < arr.length - 1; i++) {

            // Iterate from i + 1 to arr.length
            for (int j = i + 1; j < arr.length; j++) {

                // Store curr_sum = arr[i] + arr[j]
                int curr_sum = arr[i] + arr[j];

                // Check if X - curr_sum if present
                // in map
                if (map.containsKey(X - curr_sum)) {

                    // Store pair having map value
                    // X - curr_sum
                    pair p = map.get(X - curr_sum);

                    if (p.first != i && p.sec != i
                        && p.first != j && p.sec != j
                        && temp[p.first] == 0
                        && temp[p.sec] == 0 && temp[i] == 0
                        && temp[j] == 0) {

                        // Print the output
                        System.out.printf(
                            "%d,%d,%d,%d", arr[i], arr[j],
                            arr[p.first], arr[p.sec]);
                        temp[p.sec] = 1;
                        temp[i] = 1;
                        temp[j] = 1;
                        break;
                    }
                }
            }
        }
    }

    // Program for two Sum
    public static Map<Integer, pair> twoSum(int[] nums)
    {
        Map<Integer, pair> map = new HashMap<>();
```

```
        for (int i = 0; i < nums.length - 1; i++) {
            for (int j = i + 1; j < nums.length; j++) {
                map.put(nums[i] + nums[j], new pair(i, j));
            }
        }
        return map;
    }

    // to store indices of two sum pair
    public static class pair {
        int first, sec;

        public pair(int first, int sec)
        {
            this.first = first;
            this.sec = sec;
        }
    }

    // Driver Code
    public static void main(String args[])
    {
        int[] arr = { 10, 20, 30, 40, 1, 2 };
        int n = arr.length;
        int X = 91;
        Map<Integer, pair> map = twoSum(arr);

        // Function call
        fourSum(X, arr, map);
    }
}

// This code is contributed by Likhita avl.
```

Output

```
20,30,40,1
```

Complexity Analysis:

- **Time complexity:** O(n^2).
  Nested traversal is needed to store all pairs in the hash Map.
- **Auxiliary Space:** O(n^2).
  All n*(n-1) pairs are stored in hash Map so the space required is O(n^2) and the temp array takes O(n) so space comes to O(n^2).

From <https://www.geeksforgeeks.org/find-four-elements-that-sum-to-a-given-value-set-2/?ref=lbp>