# From Two Sum to Four Sum

To resolve the problems such as two sum, three sum and four sum , normally we will use the hashtable.

## 1. Two Sum

Easy

Given an array of integers, return **indices** of the two numbers such that they add up to a specific target.

You may assume that each input would have ***exactly*** one solution, and you may not use the *same* element twice.

**Example:**

Given nums = [2, 7, 11, 15], target = 9,

Because nums[**0**] + nums[**1**] = 2 + 7 = 9,

return [**0**, **1**].

### Analysis:

we can store the previously visited number in the hash table, and for any number we check if target – num[i] is in the hash table.

/// <summary>

/// Leet code 1 Two Sum

/// Given an array of integers, return indices of the two numbers such that

/// they add up to a specific target.

/// You may assume that each input would have exactly one solution.

/// Example:

/// Given nums = [2, 7, 11, 15], target = 9,

/// Because nums[0] + nums[1] = 2 + 7 = 9,

/// return[0, 1]

/// </summary>

vector<int> LeetCode::twoSum(vector<int>& nums, int target)

{

vector<int> result;

unordered\_map<int, int> num\_map;

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

{

if (num\_map.count(target - nums[i]) > 0)

{

result = { num\_map[target - nums[i]], i };

return result;

}

num\_map[nums[i]] = i;

}

return result;

}

## 15. 3Sum

Medium

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.

**Note:**

The solution set must not contain duplicate triplets.

**Example:**

Given array nums = [-1, 0, 1, 2, -1, -4],

A solution set is:

[

[-1, 0, 1],

[-1, -1, 2]

]

### Analysis:

There are two ways to do this one is sort then pick up first number and do the remaining two as two pointer scan, another way is to use hashtable to make it as a topic.

When we do the dedup and scan, one common trick is to first sort the number and then skip the duplicate number on the same position.

Please remember such trick also apply to the backtracking search, such as combination sum II

vector<vector<int>> LeetCodeHashtable::threeSum(vector<int>& nums)

{

vector<vector<int>> result;

unordered\_map<int, int> num\_map;

// sort first, so we can dedup easily

sort(nums.begin(), nums.end());

// we only need to remember last one.

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

{

num\_map[nums[i]] = i;

}

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

{

// skip all duplicated one on the same position

if ((i > 0) && (nums[i] == nums[i - 1])) continue;

for (size\_t j = i + 1; j < nums.size(); j++)

{

// skip all duplicated one on the same position

if ((j > i + 1) && (nums[j] == nums[j - 1])) continue;

int target = 0 - nums[i] - nums[j];

if (num\_map.count(target) > 0 && num\_map[target] > j)

{

result.push\_back({ nums[i], nums[j], nums[num\_map[target]] });

}

}

}

return result;

}

/// <summary>

/// LeetCode #15. 3Sum

/// </summary>

vector<vector<int>> LeetCodeHashtable::threeSumII(vector<int>& nums)

{

vector<vector<int>> result;

sort(nums.begin(), nums.end());

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

{

// skip all duplicated one on the same position

if ((i > 0) && (nums[i] == nums[i - 1])) continue;

int first = i + 1;

int last = nums.size() - 1;

while (first < last)

{

if ((first > i + 1) && (nums[first] == nums[first - 1]))

{

first++;

}

else if ((last < (int)nums.size() - 1) && (nums[last] == nums[last + 1]))

{

last--;

}

else if (nums[i] + nums[first] + nums[last] < 0)

{

first++;

}

else if (nums[i] + nums[first] + nums[last] > 0)

{

last--;

}

else

{

result.push\_back({ nums[i], nums[first], nums[last] });

first++;

last--;

}

}

}

return result;

}

## 923. 3Sum With Multiplicity

Medium

21942FavoriteShare

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

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

**Example 1:**

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

**Output:** 20

**Explanation:**

Enumerating by the values (A[i], A[j], A[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:** A = [1,1,2,2,2,2], target = 5

**Output:** 12

**Explanation:**

A[i] = 1, A[j] = A[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.

**Note:**

1. 3 <= A.length <= 3000
2. 0 <= A[i] <= 100
3. 0 <= target <= 300

### Analysis:

Although we can do it by hashtable or two pointers as the previous ones, but Let’s do it differently, using idea of backpack, an idea from DP.

We build 3 sum from all possible two sums (based on previous numbers, assuming we scan from front to end) and buid two sum from all possible one sums, the order is important to avoid duplicates.

/// <summary>

/// Leet code #923. 3Sum With Multiplicity

///

/// Given an integer array A, and an integer target, return the number of

/// tuples i, j, k such that i < j < k and A[i] + A[j] + A[k] == target.

///

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

///

//// Example 1:

///

/// Input: A = [1,1,2,2,3,3,4,4,5,5], target = 8

/// Output: 20

/// Explanation:

/// Enumerating by the values (A[i], A[j], A[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: A = [1,1,2,2,2,2], target = 5

/// Output: 12

/// Explanation:

/// A[i] = 1, A[j] = A[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.

///

///

/// Note:

///

/// 1. 3 <= A.length <= 3000

/// 2. 0 <= A[i] <= 100

/// 3. 0 <= target <= 300

/// </summary>

int LeetCode::threeSumMulti(vector<int>& A, int target)

{

int result = 0;

size\_t M = 1000000007;

vector<unordered\_map<int, int>> dp(2);

for (size\_t i = 0; i < A.size(); i++)

{

// find if any two sum add current item can form target

if (dp[1].count(target - A[i]) > 0)

{

result = (result + dp[1][target - A[i]]) % M;

}

// build new two sum with current item and existing one sum

for (auto itr : dp[0])

{

dp[1][itr.first + A[i]] =

(dp[1][itr.first + A[i]] + itr.second ) %M ;

}

// add current number to one sum

dp[0][A[i]] = (dp[0][A[i]] + 1) % M;

}

return result;

}

## 454. 4Sum II

Medium

Given four lists A, B, C, D of integer values, compute how many tuples (i, j, k, l) there are such that A[i] + B[j] + C[k] + D[l] is zero.

To make problem a bit easier, all A, B, C, D have same length of N where 0 ≤ N ≤ 500. All integers are in the range of -228 to 228 - 1 and the result is guaranteed to be at most 231 - 1.

**Example:**

**Input:**

A = [ 1, 2]

B = [-2,-1]

C = [-1, 2]

D = [ 0, 2]

**Output:**

2

**Explanation:**

The two tuples are:

1. (0, 0, 0, 1) -> A[0] + B[0] + C[0] + D[1] = 1 + (-2) + (-1) + 2 = 0

2. (1, 1, 0, 0) -> A[1] + B[1] + C[0] + D[0] = 2 + (-1) + (-1) + 0 = 0

### Analysis:

For any four sum, we build from 2 two sum, in this case we build A + B first, then build C + D, because we do not need to think about dedup in this problem, so it is reduced to a two sum problem.

/// <summary>

/// Leet code #454. 4Sum II

/// Given four lists A, B, C, D of integer values, compute how many

/// tuples (i, j, k, l)

/// there are such that A[i] + B[j] + C[k] + D[l] is zero.

///

/// To make problem a bit easier, all A, B, C, D have same length of

/// N where 0 ≤ N ≤ 500.

/// All integers are in the range of -2^28 to 2^28 - 1 and the result

/// is guaranteed to be at most 2^31 - 1.

/// Example:

/// Input:

/// A = [ 1, 2]

/// B = [-2,-1]

/// C = [-1, 2]

/// D = [ 0, 2]

///

/// Output:

/// 2

///

/// Explanation:

/// The two tuples are:

/// 1. (0, 0, 0, 1) -> A[0] + B[0] + C[0] + D[1] = 1 + (-2) + (-1) + 2 = 0

/// 2. (1, 1, 0, 0) -> A[1] + B[1] + C[0] + D[0] = 2 + (-1) + (-1) + 0 = 0

/// </summary>

int LeetCodeHashtable::fourSumCount(vector<int>& A, vector<int>& B,

vector<int>& C, vector<int>& D)

{

int result = 0;

unordered\_map<int, int> sum;

for (size\_t i = 0; i < A.size(); i++)

{

for (size\_t j = 0; j < B.size(); j++)

{

sum[A[i] + B[j]]++;

}

}

for (size\_t i = 0; i < C.size(); i++)

{

for (size\_t j = 0; j < C.size(); j++)

{

int target = 0 - C[i] - D[j];

if (sum.count(target) > 0) result += sum[target];

}

}

return result;

}

## 18. 4Sum

Medium

Given an array nums of *n* integers and an integer target, are there elements *a*, *b*, *c*, and *d* in nums such that *a* + *b* + *c* + *d* = target? Find all unique quadruplets in the array which gives the sum of target.

**Note:**

The solution set must not contain duplicate quadruplets.

**Example:**

Given array nums = [1, 0, -1, 0, -2, 2], and target = 0.

A solution set is:

[

[-1, 0, 0, 1],

[-2, -1, 1, 2],

[-2, 0, 0, 2]

]

### Analysis:

For any four sum, we build from 2 two sum, but in this case we should be careful due to the duplicate numbers, we need to build one two sum from bottom to top, and build another from top to bottom. The same dedup and early cut tricks apply. For this method, we are doing O(N^2)\*K, the K is average number duplicated two sum which hit.

There is another method is select any two number after sort and use two pointer to get another two.

/// <summary>

/// Leet code #18. 4Sum

/// Given an array S of n integers, are there elements a, b, c, and d in S

/// such that a + b + c + d = target?

/// Find all unique quadruplets in the array which gives the sum of target.

/// Note: The solution set must not contain duplicate quadruplets.

/// For example, given array S = [1, 0, -1, 0, -2, 2], and target = 0.

///

/// A solution set is:

/// [

/// [-1, 0, 0, 1],

/// [-2, -1, 1, 2],

/// [-2, 0, 0, 2]

/// ]

/// </summary>

vector<vector<int>> LeetCodeHashtable::fourSum(vector<int>& nums, int target)

{

sort(nums.begin(), nums.end());

vector<vector<int>> result;

unordered\_map<int, vector<pair<int, int>>> two\_sum;

for (int i = nums.size() - 1; i > 2; i--)

{

if (4 \* nums[i] < target) break;

// dedup

if (i < (int)nums.size() - 1 && nums[i] == nums[i + 1]) continue;

for (int j = i - 1; j > 1; j--)

{

// dedup

if (j < (i - 1) && nums[j] == nums[j + 1]) continue;

two\_sum[nums[i] + nums[j]].push\_back({ j, i });

}

}

for (int i = 0; i < (int)nums.size() - 3; i++)

{

if (4 \* nums[i] > target) break;

// dedup

if (i > 0 && nums[i] == nums[i - 1]) continue;

for (int j = i + 1; j < (int)nums.size() - 2; j++)

{

// dedup

if (j > (i + 1) && nums[j] == nums[j - 1]) continue;

int value = target - (nums[i] + nums[j]);

if (two\_sum.count(value) == 0) continue;

for (int k = two\_sum[value].size() - 1; k >= 0; k--)

{

if (two\_sum[value][k].first <= (int)j) continue;

result.push\_back({ nums[i], nums[j],

nums[two\_sum[value][k].first],

nums[two\_sum[value][k].second] });

}

}

}

return result;

}

/// <summary>

/// Leet code #18. 4Sum

/// </summary>

vector<vector<int>> LeetCodeHashtable::fourSumII(vector<int>& nums, int target)

{

sort(nums.begin(), nums.end());

vector<vector<int>> result;

for (int i = 0; i < (int)nums.size() - 3; i++)

{

if (nums[i] + nums[i+1] + nums[i+2] + nums[i+3] > target) break;

// dedup

if (i > 0 && nums[i] == nums[i - 1]) continue;

for (int j = i + 1; j < (int)nums.size() - 2; j++)

{

if (nums[i] + nums[j] + nums[j + 1] + nums[j + 2] > target) break;

// dedup

if (j > (i + 1) && nums[j] == nums[j - 1]) continue;

int value = target - (nums[i] + nums[j]);

int first = j + 1;

int last = nums.size() - 1;

while (first < last)

{

if ((nums[first] + nums[first + 1] > value) ||

(nums[last - 1] + nums[last] < value))

{

break;

}

if ((first > j + 1) && (nums[first] == nums[first - 1]))

{

first++;

}

else if ((last < nums.size() - 1) &&

(nums[last] == nums[last + 1]))

{

last--;

}

else if (nums[first] + nums[last] < value)

{

first++;

}

else if (nums[first] + nums[last] > value)

{

last--;

}

else

{

result.push\_back({ nums[i], nums[j], nums[first],

nums[last] });

first++;

last--;

}

}

}

}

return result;

}