**Algorithm Notes**

目录

[1 Array 1](#_Toc2607978)

[1.1 HashTable 1](#_Toc2607979)

[1 Two Sum 1](#_Toc2607980)

[1.2 Two Pointer 1](#_Toc2607981)

[11 container with most water 2](#_Toc2607982)

[15 3 sum 3](#_Toc2607983)

[16 3 sum closest 3](#_Toc2607984)

[18 4 sum 3](#_Toc2607985)

[1.3 backtracking 3](#_Toc2607986)

[39 combination sum 4](#_Toc2607987)

[40 Combination sum II 4](#_Toc2607988)

[1.4 Binary Search 4](#_Toc2607989)

[4 Median of two sorted array 5](#_Toc2607990)

[454 4sum II 6](#_Toc2607991)

# 1 Array

## 1.1 HashTable

利用hash数据结构记录数组中元素的出现，然后对相应的元素直接检查获取。

### 1 Two Sum

通过hash表记录各个元素的出现，然后在遍历元素的过程中，检查目标和与该元素的差是否出现，**值得注意的是：相对于完整的遍历存储好再检查，可以边遍历边检查。**

## 1.2 Two Pointer

在**有序**的前提下寻找符合特定条件的两个元素时，相对于暴力的两层遍历叠加，可以使用一头一尾两个指针，分别从前后逼近查找。

**核心代码：**

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

int res = target-nums[i];

if(num\_map.find(res)!=num\_map.end()){

result[0] = num\_map[res];

result[1] = i;

return result;

}else{

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

}

}

### 11 container with most water

Given *n* non-negative integers *a1*, *a2*, ..., *an*, where each represents a point at coordinate (*i*, *ai*).  Find two lines, which together with x-axis forms a container, such that the container contains the most water.

**Example:**

**Input:** [1,8,6,2,5,4,8,3,7]

**Output:** 49

核心代码：

int maxArea(vector<int>& height) {

int l = 0, r = height.size()-1;

int max\_area = min(height[l],height[r])\*(r-l);

while(l<r){

if(height[l]<height[r])l++;

else r--;

max\_area = max(max\_area,min(height[l],height[r])\*(r-l));

}

return max\_area;

}

思路：

两个指针分别从两侧往中间寻找，每次移动**较小**的值对应的指针。为什么较小？嘻嘻

### 15 3 sum

三个数的求和，指定其中一个（依序便利），对另外两个数使用两个指针

核心代码：

while(l<r){

if(nums[l]+nums[r]<-nums[i])l++;

else if(nums[l]+nums[r]>-nums[i])r--;

else{

tmp[0] = nums[i];

tmp[1] = nums[l];

tmp[2] = nums[r];

res.push\_back(tmp);

while(l<r && nums[l]==tmp[1])l++;

while(l<r && nums[r]==tmp[2])r--;

}

}

### 16 3 sum closest

同15，不同的地方在于最近的元素需要记录最小的差

### 18 4 sum

同15，只是外面有两层遍历，相应的需要两层去重

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

…

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

## 1.3 backtracking

对每个可能的分支遍历时，达到一定条件便回溯到之前的位置继续遍历其他分支。值得注意的是：为了节省空间和时间，结果集可以在遍历的过程中通过函数参数传递从而达到只使用一个结果集的目的。

**关键点：**

* **参数传递**
* **有进有出**

### 39 combination sum

之所以使用回溯是因为允许加和的元素没有限制，所以需要放开空间寻找，达到条件再回溯。

**Input:** candidates = [2,3,6,7], target = 7,

**A solution set is:**

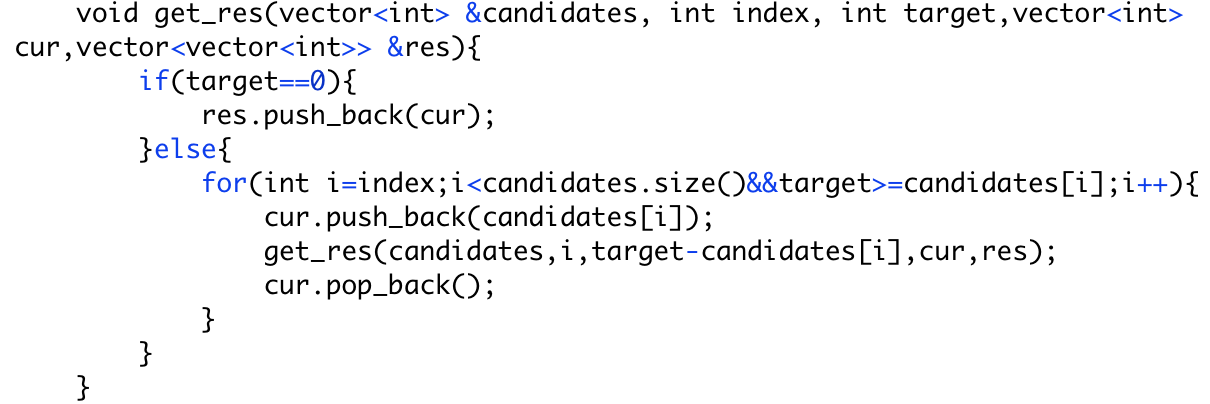
[

[7],

[2,2,3]

]

核心代码：



**注意：**

* **cur有push\_back也有pop\_back, 满足回溯要求的有进有出**
* **为了减少不必要的操作（如减去大于target的candidates元素），可以提前判断（在循环中），减少时间复杂度。**

### 40 Combination sum II

同39

## 1.4 Binary Search

在search过程中，将待search的数列两两分开处理，减少时间复杂度。

### 4 Median of two sorted array

Find the median of the two sorted arrays. The overall run time complexity should be O(log (m+n)).

**Example 1:**

nums1 = [1, 3]

nums2 = [2]

The median is 2.0

如何使用二分搜索呢，因为有序，可以理解为两个有序数列的中位数会把两个数列分别分为左半部分和右半部分，各自的左半部分的长度和等于右半部分的长度和，且**左半部分的最大值大于右半部分的最小值（又因为数列是有序的，所以只用交叉比较）**，那么如何找到切分两个数列的点，就需要用到二分搜索，这里的二分不是针对于数列的长度，而是中位数（也可以是其他顺序统计量，实际上除了第一次是中位数，后续查找时都不是中位数）的二分，中位数二分为数列1的左半部分和数列2的左半部分，然后比较临界值是否满足上述标黑的性质，根据不同情况截取掉肯定不包含中位数（顺序统计量的）的部分，然后在剩余的数列中继续寻找，注意中位数要减去截取掉的数列的长度，变成一般的顺序统计量。

核心代码：

double get\_median(vector<int> &nums1,vector<int> &nums2,int l,int r,int k,int flag){

if(nums1.size()-l>nums2.size()-r)

if(l==nums1.size()){

}else if(k==1){

}else{

int lk = min(int(nums1.size()-l),k/2);

int res\_k = k-lk;

if(nums1[l+lk-1]<nums2[r+res\_k-1])

return get\_median(nums1,nums2,l+lk,r,k-lk,flag);

else

return get\_median(nums1,nums2,l,r+res\_k,k-res\_k,flag);

}

return 0.0;

}

### 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.

**Input:**

A = [ 1, 2]

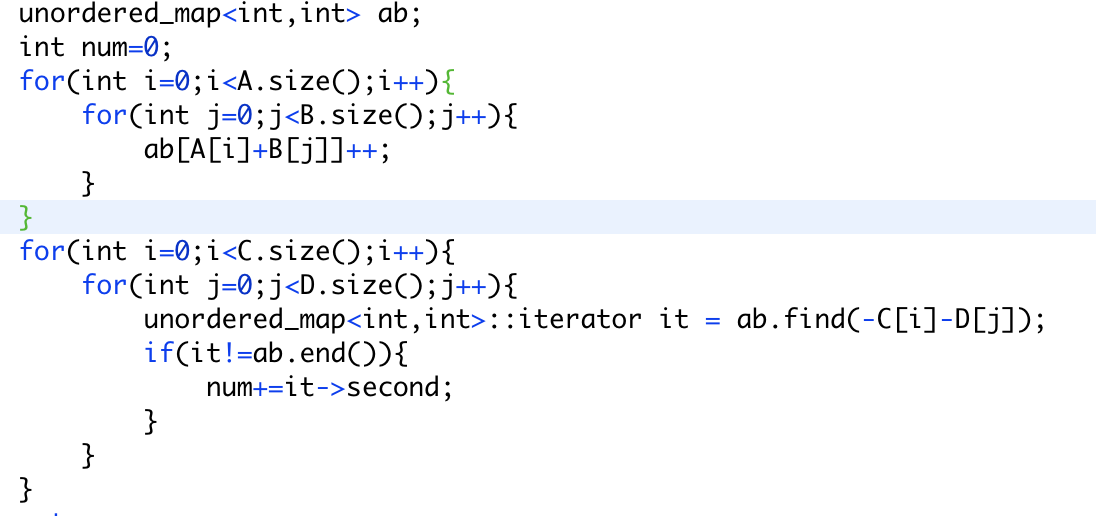
B = [-2,-1]

C = [-1, 2]

D = [ 0, 2]

不同于一般的binary search是针对于一个数列，这个是针对于多个数列，二分的含义从将一个数列分为两个数列处理变为将多个数列划分为两堆数列，依次划分到只剩两个，然后统计结果。当然，在计算的过程中，是自底向上的。

当然，这个题目要求加和为0，核心要加和，所以在统计到只有两个数列时，要两两加和，并标记加和后结果的出现次数。



**PS:** 1 不需要有序的前提下，使用unordered\_map比map快很多

2 学会利用map.find已查找的iterator，不要重复查找