# 题目

给定一个整数数组nums和一个目标值target，请你在该数组中找出和为目标值的那两个整数，并返回他们的数组下标。

你可以假设每种输入只会对应一个答案。但是，你不能重复利用这个数组中同样的元素。

示例:

给定 nums = [2, 7, 11, 15], target = 9

因为 nums[0] + nums[1] = 2 + 7 = 9

所以返回 [0, 1]

提示：

2 <= nums.length <= 10^4

-10^9 <= nums[i] <= 10^9

-10^9 <= target <= 10^9

只会存在一个有效答案

进阶：你可以想出一个时间复杂度小于 O(n2) 的算法吗？

**类似题目：面试题57**

# 分析

## 方法一：暴力破解

**思路：**

**代码：**

class Solution {

public:

vector<int> twoSum(vector<int>& nums, int target) {

int n = nums.size();

for (int i = 0; i < n; ++i) {

for (int j = i + 1; j < n; ++j) {

if (nums[i] + nums[j] == target) {

return {i, j};

}

}

}

return {};

}

};

## 方法二：哈希表[一遍]

**思路：**

将和差运算，转换为在unordered\_map中查找目标值的问题。

**代码：**

class Solution {

public:

vector<int> twoSum(vector<int>& nums, int target) {

unordered\_map<int, int> numToIndex;

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

auto it = numToIndex.find(target - nums[i]);

if (it != numToIndex.end()) {

return {it->second, i};

}

numToIndex[nums[i]] = i;

}

return {};

}

};

这个实现中，我们首先遍历数组，将每个元素的值和对应的下标存储在哈希表 numToIndex 中。在遍历过程中，我们检查当前元素的补数（即 target - nums[i]）是否在哈希表中，如果存在，则说明找到了答案，返回两个元素的下标。如果不存在，则将当前元素存储到哈希表中。这样可以保证不会重复使用相同的元素。

或：

class Solution {

public:

vector<int> twoSum(vector<int>& nums, int target) {

unordered\_map<int,int> record;

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

int complement = target - nums[i];

if(record.find(complement) != record.end()){

int res[] = {i, record[complement]};

return vector<int>(res, res + 2);

}

record[nums[i]] = i;

}

}

};

## 方法三：哈希表[两遍]

**思路：**

为了对运行时间复杂度进行优化，我们需要一种更有效的方法来检查数组中是否存在目标元素。如果存在，我们需要找出它的索引。保持数组中的每个元素与其索引相互对应的最好方法是什么？哈希表。

通过以空间换取速度的方式，我们可以将查找时间从O(n) 降低到O(1)。哈希表正是为此目的而构建的，它支持以近似恒定的时间进行快速查找。我用“近似”来描述，是因为一旦出现冲突，查找用时可能会退化到O(n)。但只要你仔细地挑选哈希函数，在哈希表中进行查找的用时应当被摊销为O(1)。

一个简单的实现使用了两次迭代。在第一次迭代中，我们将每个元素的值和它的索引添加到表中。然后，在第二次迭代中，我们将检查每个元素所对应的目标元素（target - nums[i]）是否存在于表中。注意，该目标元素不能是nums[i] 本身！

**代码：**

class Solution {

public:

vector<int> twoSum(vector<int>& nums, int target) {

unordered\_map<int,int> m;

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

m[nums[i]] = i; //向map中添加元素

vector<int> res;

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

{

if(m.find(target-nums[j]) != m.end() && m[target-nums[j]] != j) //如果m中存在对应的键值，且不为i

{

res.push\_back(j);

res.push\_back(mp[target-nums.at(j)]);

return res;

}

}

return res;

}

};

**复杂度：**

时间复杂度：O(n)

我们把包含有n个元素的列表遍历两次。由于哈希表将查找时间缩短到O(1)，所以时间复杂度为O(n)。

空间复杂度：O(n)

所需的额外空间取决于哈希表中存储的元素数量，该表中存储了n个元素。