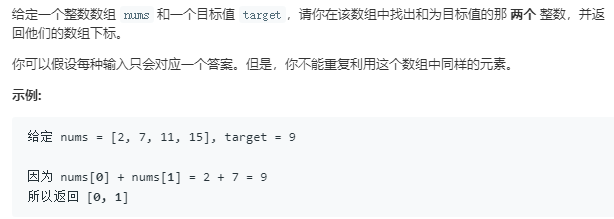
# Ltc01算法题

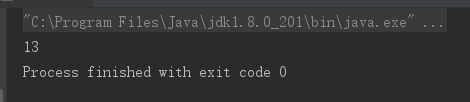
题目描述



我的答案

package com.lagoon.test;  
  
import java.util.HashMap;  
import java.util.Map;  
  
*/\*\*  
 \** ***@Author*** *WinkiLee  
 \** ***@Date*** *2019/5/6 15:43  
 \** ***@Description*** *\*/*public class Solution {  
  
 public int[] twoSum(int[] nums, int target) {  
  
 Map<Integer,Integer> integerMap=new HashMap<>();  
 for (int i=0;i<nums.length;i++){  
 if (integerMap.containsKey(nums[i])){  
 return new int[]{integerMap.get(nums[i]),i};  
 }  
 integerMap.put(target-nums[i],i);  
 }  
 return null;  
 }  
  
 public static void main(String[] args) {  
 int[] nums={15,7,11,2};  
 int target=9;  
 Solution solution=new Solution();  
 int[] x=solution.twoSum(nums,target);  
 for (int i=0;i<x.length;i++){  
 System.*out*.print(x[i]);  
 }  
 }  
  
}

运行截图



测试通过

流程分析

测试数组

int[] nums={15,7,11,2};

目标值

int target=9;

1. 首先定义一个HashMap类型的数据integerMap，此时里面没有任何键值对

Map<Integer,Integer> integerMap=new HashMap<>();

1. 进入数组循环

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

1. if条件语句判断，当前hashmap中所有键里是否有当前数组元素值，有则返回一个数组，数组中的元素值分别为这个键和当前数组元素下标；没有则向hashmap里添加键值对，键为目标值减去数组元素值，键值对的值为数组元素下标

* i=0 nums[i]=15 不满足if语句

所以执行

integerMap.put(target-nums[i],i);

hashmap中存入键值对（-6,0）

* i=1 nums[i]=7 不满足if条件语句

所以执行

integerMap.put(target-nums[i],i);

hashmap中存入键值对（2,1）

* i=2 nums[i]=11 不满足if语句

所以执行

integerMap.put(target-nums[i],i);

hashmap中存入键值对（-2,2）

* i=3 nums[i]=2 满足if语句

所以执行

return new int[]{integerMap.get(nums[i]),i};

返回一个数组，数组中的两个元素为{1,3}

其他题解

**方法一：暴力法**

暴力法很简单，遍历每个元素x，并查找是否存在一个值与target-x相等的目标元素

Demo：  
package com.lagoon.test;  
  
*/\*\*  
 \** ***@Author*** *WinkiLee  
 \** ***@Date*** *2019/5/6 16:02  
 \** ***@Description*** *暴力法  
 \*/*public class Solution2 {  
  
 public int[] twoSum(int[] nums,int target){  
 for (int i=0;i<nums.length;i++){  
 for (int j=i+1;j<nums.length;j++){  
 if (nums[j]==target-nums[i]){  
 return new int[] {i,j};  
 }  
 }  
 }  
 throw new IllegalArgumentException("No two sum solution");  
 }  
  
 public static void main(String[] args) {  
 int[] nums={15,7,11,2};  
 int target=9;  
 Solution solution=new Solution();  
 int[] x=solution.twoSum(nums,target);  
 for (int i=0;i<x.length;i++){  
 System.*out*.print(x[i]);  
 }  
 }  
}

复杂度分析

时间复杂度：O(n2),对于每个元素，我们试图通过遍历数组的其余部分来寻找它所对应的的目标元素，这将耗费O(n)的时间，因此时间复杂度为O(n2)

空间复杂度O(1)

**方法二：两遍哈希表**

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书否存在目标元素，如果存在，我们需要找到它的索引，保持数组中每个元素与其索引相互对应的最好方法就是哈希表

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，将内个元素的值和它的索引添加到表中，然后，在第二次迭代中，将检查每个元素所对应的的目标元素（target-nums[i]）是否存在表中。注意：该目标元素不能是nums[i]本身

Demo

package com.lagoon.test;  
  
import java.util.HashMap;  
import java.util.Map;  
  
*/\*\*  
 \** ***@Author*** *WinkiLee  
 \** ***@Date*** *2019/5/6 16:16  
 \** ***@Description*** *两遍哈希表  
 \*/*public class Solution3 {  
  
 public int[] twoSum(int[] nums,int target){  
 Map<Integer,Integer> map=new HashMap<>();  
 for (int i=0;i<nums.length;i++){  
 map.put(nums[i],i);  
 }  
 for (int i=0;i<nums.length;i++){  
 int complement=target-nums[i];  
 if (map.containsKey(complement)&&map.get(complement)!=i){  
 return new int[]{i,map.get(complement)};  
 }  
 }  
 throw new IllegalArgumentException("No two sum solution");  
 }  
  
 public static void main(String[] args) {  
 int[] nums={15,7,11,2};  
 int target=18;  
 Solution solution=new Solution();  
 int[] x=solution.twoSum(nums,target);  
 for (int i=0;i<x.length;i++){  
 System.*out*.print(x[i]);  
 }  
 }  
}

复杂度分析：

时间复杂度：O(n)，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

**方法三：一遍哈希表**

事实证明，可以一次完成。在进行迭代并将元素插入到表中的同时，回过头来检查表中是否已经存在当前元素所对应的目标元素，如果存在，即找到了对应解，并立即将其返回

Demo：

package com.lagoon.test;  
  
import java.util.HashMap;  
import java.util.Map;  
  
*/\*\*  
 \** ***@Author*** *WinkiLee  
 \** ***@Date*** *2019/5/6 16:28  
 \** ***@Description*** *一遍哈希表  
 \*/*public class Solution4 {  
 public int[] twoSum(int[] nums, int target) {  
 Map<Integer, Integer> map = new HashMap<>();  
 for (int i = 0; i < nums.length; i++) {  
 int complement = target - nums[i];  
 if (map.containsKey(complement)) {  
 return new int[] { map.get(complement), i };  
 }  
 map.put(nums[i], i);  
 }  
 throw new IllegalArgumentException("No two sum solution");  
 }  
  
 public static void main(String[] args) {  
 int[] nums={15,7,11,2};  
 int target=9;  
 Solution solution=new Solution();  
 int[] x=solution.twoSum(nums,target);  
 for (int i=0;i<x.length;i++){  
 System.*out*.print(x[i]);  
 }  
 }  
}

时间复杂度：O(n)， 只遍历了包含有 n 个元素的列表一次。在表中进行的每次查找只花费 O(1) 的时间。

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，该表最多需要存储 n 个元素。