简十三 已于 2022-06-27 12:19:29 修改 1612 收藏 9 版权

分类专栏： LeetCode讲解(看不懂打我系列) 文章标签： leetcode 算法 职场和发展 c语言

题目要求

一个包含 n 个整数的数组 nums，判断 nums 中是否存在三个元素 a，b，c，使得 a + b + c = 0 ？请出所有和为 0 且不重复的三元组 。

：答案中不可以包含重复的三元组。

1：

：nums = [-1,0,1,2,-1,-4]

：[[-1,-1,2],[-1,0,1]]

2：

：nums = []

：[]

3：

：nums = [0]

：[]

：

nums.length <= 3000

<= nums[i] <= 105

：力扣（LeetCode）

一看是不是很熟悉？没错，我的之前写了一篇 **“两数之和”** 的博客，跟这个题很相似，但是做法却更难一**这篇博客前建议大家可以参考之前的那一篇博客加深一下理解。**

log.csdn.net/m0_63988748/article/details/125255208（最多盛水问题——双指针）

题目理解以及思路分析

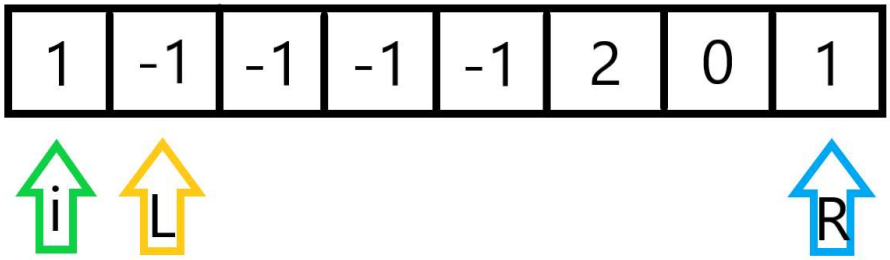
理解：

就是从给出的数组中找到符合要求的**三个数字**，然后组成一个**二维数组**

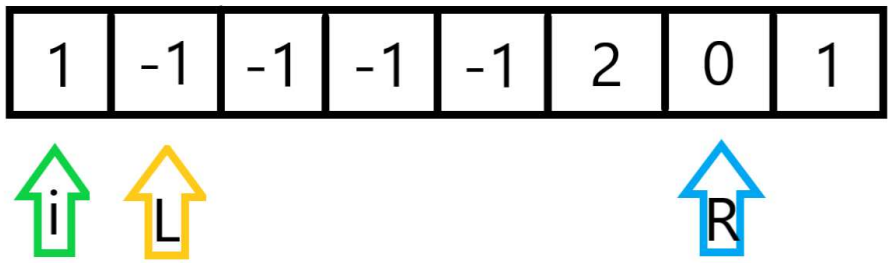
分析：

：

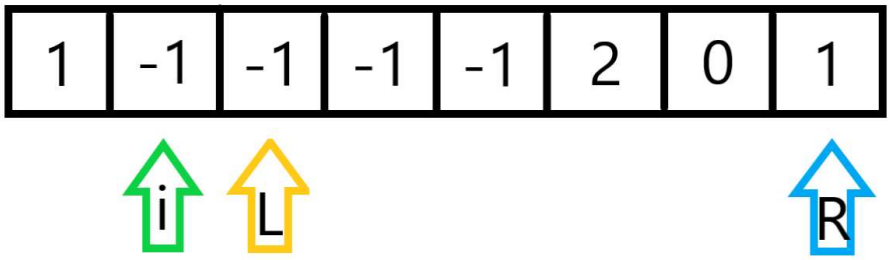
三个数字，我们先定义一个 “双指针”，但是这两个指针的位置不是随便的。



指针分别为 **左指针——** L ， **右指针——** R 。由图可以看出，指针初始的位置就是这样。那我们如何进行要知道一个好的遍历方法会让问题事半功倍。



的 **sum = i + L + R = 1** ，可见 **sum > 0** 。因此我们让 **右指针左移**，然后接着判断这一次的 sum 的 **sum < 0** ，则 左指针右移，直至左右指针重合。然后 i + 1，开始下一轮的遍历。



下是对这个题目有了更清晰的理解和认识?

:

明白了上述的思路，接下来我们就要进行的是如何让遍历更加的简便。

1.给数组排序，使其按照从小到大的顺序排列。

2.对于 nums = NULL 以及numsSize < 3 的全部返回 NULL。

3.对于排过序的数组，凡是 nums [i] = 0 的，全部返回 0.

:

下面就只剩最后一个问题了——**去重**。这一个我们结合具体的代码进行讲解。

代码分部讲解

部分:

```
/排序，使其代码按照从低到高的顺序排列
nt inc(const void*a,const void*b)
 {
   return* (int*)a - * (int*)b;
 }

nt** threeSum(int* nums, int numsSize, int* returnSize, int** returnColumnSizes){
   *returnSize = 0;
   int i;
   if(nums == NULL||numsSize < 3)  //根据题意可以先排除一部分
     return NULL;

   qsort(nums,numsSize,sizeof(int),inc);  //升序排列
```

释:

nc (const void* a, const void* b)

turn* (int*) a - * (int*) b;

是C语言中的排序函数，其中 inc 全拼是 increase（增加），代表着 “升序排列”，这个名字也可以取。
t (nums, numsSize, sizeof(int), inc);

. 第一个参数为待排序数组首地址。

. 第二个参数为数组长度（常用sizeof [a [n]]）来代表

. 第三个参数为数组元素所占字节

. 第四个参数为所调用函数的指针，函数名即是函数的指针，可直接写函数名，调用函数用来确定排序的方式

部分：

```
//分配返回数组的列数
*returnColumnSizes = (int*)malloc(sizeof(int)*numsSize* numsSize);
//分配返回数组
int** han = (int**)malloc(sizeof(int*)*numsSize* numsSize);
for(i=0;i<numsSize;i++)
  {
    if(nums[i]>0)  //对于排过序的数组来说，如果nums[i] = 0，则三数之和不可能为 0
      break;
    if(i>0 && nums[i]==nums[i-1])  //如果与上一个数相同则跳过（去重）
      continue;
```

分配返回数组的列数
eturnColumnSizes = (int*) malloc (sizeof(int*)* numsSize* numsSize);

分配返回数组
t** han = (int**) malloc (sizeof(int*)* numsSize* numsSize);
是“动态分配”，malloc函数。为其分配一个动态的空间。

部分：

```
int L = i+1, R = numsSize-1;  //定义左右指针
while(L < R) //循环
  {
    int sum = nums[i] + nums[L] + nums[R];  //三数之和
    int n,m;
    if(sum == 0) //如果 三数之和为零
      {
        han[*returnSize] = (int*)malloc(sizeof(int)*3);//为二维数组的行数分配三个空间
        han[*returnSize][0] = nums[i];
        han[*returnSize][1] = nums[L];
        han[*returnSize][2] = nums[R];
        (*returnColumnSizes)[*returnSize] = 3; //返回数组当前行的列数为3
        *returnSize+=1; //行数加 1
        //左右去重
        do{
          n=L++;
        }while(n<R && nums[L] == nums[n]);
        do{
          m=R--;
        }while(L<m && nums[R] == nums[m]);
      }
    else if(sum < 0)
      L++;
    else if(sum > 0)
      R--;
```

```
return han;  //返回数组
```

解：

returnSize] = (int)malloc(sizeof(int)*3);//为二维数组的行数分配三个空间
*returnSize][0] = nums[i];
*returnSize][1] = nums[L];
*returnSize][2] = nums[R];
urnColumnSizes)[*returnSize] = 3; //返回数组当前行的列数为3
urnSize+=1; //行数加 1
部分跟之前的“两数之和”里面很相似，所以就不过多的讲解
需要注意的一点是：

```
returnSize+=1
```

以写成

```
returnSize++
```

，优先级的问题会导致代码出现错误。
可以将其改为

```
*returnSize)++
```

```
do{
   n=L++;
}while(n<R && nums[L] == nums[n]);

do{
   m=R--;
}while(L<m && nums[R] == nums[m]);
```

的目的就是，当sum == 0 的时候，对左右指针下一次的移动进行去重。
，很多人会选择写另一种代码程序：

```
while(L < R && nums[L] == nums[++L]);

while(L < R && nums[R] == nums[--R]);
```

写法是非常不规范的，但是用起来确实极度的舒适；
讲解一下这串代码
（xxxxxxx）；
加了一个 “；”，就意味着，当里面的条件为 “真” 时，部执行其下面的代码，当为 “非” 时，便会去执

，上述的代码意思便可以解释为，令原本的 nums[L] 和 nums[R] 的值等于其右/左一个的值，以此类至不符合条件结束，这就达到了左右去重的目的。

完整的代码：

```
* The sizes of the arrays are returned as *returnColumnSizes array.

ote: Both returned array and *columnSizes array must be malloced, assume caller calls free().
*/ 8 | int inc(const void*a,const void*b)
 {
   return* (int*)a - * (int*)b;
 }
nt** threeSum(int* nums, int numsSize, int* returnSize, int** returnColumnSizes){
   *returnSize = 0;
   int i;
   if(nums == NULL||numsSize < 3)
     return NULL;
   qsort(nums,numsSize,sizeof(int),inc);
   *returnColumnSizes = (int*)malloc(sizeof(int)*numsSize* numsSize);
   int** han = (int**)malloc(sizeof(int*)*numsSize* numsSize);
   for(i=0;i<numsSize;i++)
     {
       if(nums[i]>0)
         break;
       if(i>0 && nums[i]==nums[i-1])
         continue;
       int L = i+1, R = numsSize-1;
       while(L < R)
         {
           int sum = nums[i] + nums[L] + nums[R];
           int n,m;
           if(sum == 0)
             {
               han[*returnSize] = (int*)malloc(sizeof(int)*3);
               han[*returnSize][0] = nums[i];
               han[*returnSize][1] = nums[L];
               han[*returnSize][2] = nums[R];
               (*returnColumnSizes)[*returnSize] = 3;
               *returnSize+=1;
               //(*returnSize)++;

               /*非常不规范的写法，但是极其舒适！
               while(L < R && nums[L] == nums[++L]);
               while(L < R && nums[R] == nums[--R]);
               */
               do{
                 n=L++;
               }while(n<R && nums[L] == nums[n]);
               do{
                 m=R--;
               }while(L<m && nums[R] == nums[m]);
             }
           else if(sum < 0)
             L++;
           else if(sum > 0)
             R--;
         }
     }
   return han;
```