

数组与数组列表

Array & ArrayList

无隅

什么是数组

- 数组由相同类型的元素（element）的集合所组成的结构
- 分配一块连续的内存来存储元素
- 利用元素的索引（index）可以计算出该元素对应的储存地址

数组的特性

- 在内存中为连续空间

定址公式: $\text{addr}(\text{curElem}) = \text{addr}(\text{intialElem}) + \text{sizeof}(\text{curElem}) * \text{index}$

- 存储在数组中的元素是相同类型的
- 通过index获取数组元素的时间复杂度为 $O(1)$

`sizeof(int)` 就是告诉int的大小（占的位数）
16位系统是2，32位系统是4

Java中如何表示数组

例1: `int[] a = new int[10];`

例2:

`int[][] nums= new int[3][];`

`nums[0] = new int[2];`

`nums[1] = new int[5];`

`nums[2] = new int[3];`

为啥用数组

- 简化代码， 提高效率
- 将相同类型的元素组织在一起

数组操作的局限性

- 数组基本操作：
 1. 通过下标获取值
 2. 获取长度
- 需要更多的基础操作
 - 添加元素
 - 删除元素
 - 查找元素
 - 等等.....

数组典型面试题目

两数之和

给定一个整数数组（无重复元素）和一个目标值，找出数组中和为目标值的两个数。

按照从小到大的顺序输出结果对
可以假设每个输入只对应一种答案

例：

Input: numbers={2, 7, 11, 15}, target=9

Output: {2, 7}

```
public int[] twoSum(int[] nums, int target) {  
    //Todo: implement this function.  
}
```


思路1: brute force

- 暴力遍历：遍历取一个数，计算它与其它数字之和，
遍历全部情况得到想要的结果对
- 时间复杂度： $O(n^2)$

```
int[] twoSum(int[] nums, int target) {  
    int[] result = new int[2];  
    if (nums.length < 2) {  
        return result;  
    }  
    for (int i = 0; i < nums.length-1; i++) {  
        for (int j = i+1; j < nums.length; j++) {  
            if (nums[i] + nums[j] == target) {  
                if (nums[i] < nums[j]) {  
                    result[0] = nums[i];  
                    result[1] = nums[j];  
                } else {  
                    result[0] = nums[j];  
                    result[1] = nums[i];  
                }  
                return result;  
            }  
        }  
    }  
    return result;  
}
```

思路1总结

- 时间复杂度: $O(n^2)$
- brute force 暴力解法
- 存在无意义的操作

例:

nums : {1, 4, 20, 15, 8, 6, 3 }, target = 10.

第一次循环: $1 + 20 > 10$

之后的循环: $4 + 20$, 没有必要

思路2：排序 + 两根指针

- 通过对数组排序与两根指针组合，减少无意义的遍历
- 两根指针：排序后， 一根指针（start指针）指向数组第一个元素（数组中最小元素）， 另一个指针（end指针）指向数组最后一个元素（数组中最大元素）
- 核心想法： 如果现在两根指针所指元素之和大于目标值， 则表明现在两数之和过大， 应使end指针指向更小的数， 即索引减小（end--）， 反之则表明现在两数之和过小， 应使start指针指向更大的数， 即索引增加（start++）

```
int[] twoSum(int[] nums, int target) {  
    int[] result = new int[2];  
    Arrays.sort(nums);  
    int start = 0, end = nums.length - 1;  
    while (start < end) {  
        if (nums[start] + nums[end] == target) {  
            result[0] = nums[start];  
            result[1] = nums[end];  
            return result;  
        }  
        if (nums[start] + nums[end] > target) {  
            end--;  
        }  
        if (nums[start] + nums[end] < target) {  
            start++;  
        }  
    }  
    return result;  
}
```

思路2总结

- 通过对数组排序与两根指针组合，减少无意义的遍历
- 使用两个指针（而不是一个）以相同/相反的方向遍历数组
 - 一根指针（start指针）指向数组第一个元素（数组中最小元素），
 - 另一个指针（end指针）指向数组最后一个元素（数组中最大元素）

if sum == target, get result

if sum > target, end--.

if sum < target, start++.

- 时间复杂度分析：排序： $O(n\log n)$ ，两根指针算法： $O(n)$
时间复杂度： $O(n\log n) + O(n) = O(n\log n)$

三数之和

给定一个包含 n 个整数的数组（无重复元素）`nums` 和一个目标值 `target`，找出数组中和为目标值的三个数。

可以假设每个输入只对应一种答案

例如，

给定数组 `nums = [-1, 0, 1, 2, -4]`，`target = 0`

满足要求的三元组集合为： `[-1, 0, 1]`

```
public int[] threeSum(int[] nums, int target) {  
    // Todo: implement this function.  
}
```

思路

- 当然可以用暴力遍历求解，时间复杂度为 $O(n^3)$
- 看看排序 + 两根指针算法是否能够求解

遍历第一个数字num1，看看另外两数之和是否能满足 $target - num1$ ，这就转化为两数之和的问题

- 时间复杂度：
 1. 对于不同的n个第一个数：排序 + 两数之和
 2. $O(n \log n) + n * O(n) = O(n^2)$


```
int[] threeSum(int[] nums, int target) {  
    int[] result = new int[3];  
    if (nums.length < 3) {  
        return nums;  
    }  
    Arrays.sort(nums);  
    for (int i = 0; i < nums.length-2; i++) {  
        int first = i+1, second = nums.length-1, new_target = target-nums[i];  
        while (first < second) {  
            if (nums[first] + nums[second] == new_target) {  
                result[0] = nums[i];  
                result[1] = nums[first];  
                result[2] = nums[second];  
                return result;  
            }  
            if (nums[first] + nums[second] > new_target) {  
                second--;  
            }  
            if (nums[first] + nums[second] < new_target) {  
                first++;  
            }  
        }  
    }  
    return result;  
}
```

K-Sum解法总结

- 排序
- 尝试遍历第一个数，将问题转化为 $k - 1$ Sum
- 时间复杂度：

$$2\text{-Sum: } O(n \log n) + O(n) = O(n \log n)$$

$$3\text{-Sum: } O(n \log n) + O(n^2) = O(n^2)$$

$$4\text{-Sum: } O(n \log n) + O(n^3) = O(n^3)$$

$$k\text{-Sum: } O(n \log n) + O(n^{k-1}) = O(n^{k-1})$$

反转数组

给定一个数组，反转数组中的所有数字
例：

Input: {1, 2, 3, 4, 5, 6, 7}

Output: {7, 6, 5, 4, 3, 2, 1}

```
public void reverseArray(int[] nums) {  
    // TODO: implement this function.  
    int start = 0, end = nums.length - 1;  
    while (start < end) {  
        swap(nums, start++, end--);  
    }  
}  
  
private void swap(int[] nums, int first, int second) {  
    int temp = nums[first];  
    nums[first] = nums[second];  
    nums[second] = temp;  
}
```

奇数偶数排序

给定一组整数，对它们进行排序，以便所有奇数整数在偶数整数之前出现。元素的顺序可以改变。排序的奇数和偶数的顺序无关紧要。

例：

Input: {4, 3, 5, 2, 1, 11, 0,

Output: {9, 3, 5, 11, 1, 2, 0,

```
public void oddEvenSort(int[] nums) {  
    int first = 0, second = nums.length - 1;  
    while (first < second) {  
        while (first < second && nums[first] % 2 == 1) {  
            first++;  
        }  
        while (first < second && nums[second] % 2 == 0) {  
            second--;  
        }  
        if (first < second) {  
            swap(nums, first++, second--);  
        }  
    }  
}
```

合并两个有序数组

给定两个有序整数数组 `nums1` 和 `nums2`，
请按递增顺序将它们合并到一个排序数组中
例：

Input: {1, 3, 5}, {2, 4, 6}

Output: {1, 2, 3, 4, 5, 6}

```
public int[] merge(int[] arr1, int[] arr2) {  
    int[] result = new int[arr1.length + arr2.length];  
    int index = 0, index1 = 0, index2 = 0;  
    while (index1 < arr1.length && index2 < arr2.length) {  
        if (arr1[index1] < arr2[index2]) {  
            result[index++] = arr1[index1++];  
        } else {  
            result[index++] = arr2[index2++];  
        }  
    }  
    for (int i = index1; i < arr1.length; i++) {  
        result[index++] = arr1[i];  
    }  
    for (int i = index2; i < arr2.length; i++) {  
        result[index++] = arr2[i];  
    }  
    return result;  
}
```

数组列表ArrayList

- Java中的声明: `ArrayList<Integer> list = new ArrayList<Integer>();`
- 基本操作

方法	输入	输出	时间复杂度
Get	index	value	$O(1)$
Set	index, value	void	$O(1)$
Add	index, value	void	$O(n)$
Remove	index/value	void	$O(n)$
Find	value	Boolean	$O(n)$

ArrayList的实现

- 定义属性字段

1. 在数组的基础上实现：存储数据信息 `int data[]`
2. 属性： `data`, `size`, `capacity`
3. 构造器实现

- 定义方法

1. `add`, `get`, `set`, `remove`

ArrayList的实现 - 属性及构造器

```
public class ArrayList{  
  
    private int capacity;  
    private int size;  
    private int[] data;  
  
    public ArrayList(int capacity) {  
        this.capacity = capacity;  
        this.size = 0;  
        this.data = new int[capacity];  
    }  
  
}
```

ArrayList的实现 - 方法及函数

```
public class ArrayList{  
    public int get(int index) {  
        // Todo: implement this method.  
    }  
    public void set(int index, int value) {  
        // Todo: implement this method.  
    }  
    public void add(int value) {  
    }  
    public void add(int index, int value) {  
        // Todo: implement this method.  
    }  
    public void remove(int index) {  
        // Todo: implement this method.  
    }  
    public void remove(int value) {  
    }  
}
```

ArrayList的实现 - Get方法与Set方法

```
public int get(int index) {  
    return data[index];  
}
```

```
public int get(int index) {  
    if (index < 0 || index >= size) {  
        // throw Exception  
    }  
    return data[index];  
}
```

```
public void set(int index, int value) {  
    if (index < 0 || index >= size) {  
        // throw Exception  
    }  
    data[index] = value;  
}
```

ArrayList的实现 - Add方法

```
public void add(int index, int value) {  
    if (index < 0 || index > size) {  
        // throw Exception  
    }  
    size++;  
    for (int i = size - 1; i >= index + 1; i--) {  
        data[i] = data[i - 1];  
    }  
    data[index] = value;  
}
```

```
public void add(int index, int value) {  
    if (index < 0 || index > size) {  
        // throw Exception  
    }  
    if (size == capacity) {  
        resize();  
    }  
    size++;  
    for (int i = size - 1; i >= index + 1; i--) {  
        data[i] = data[i - 1];  
    }  
    data[index] = value;  
}  
  
private void resize() {  
    capacity *= 2;  
    int[] new_data = new int[capacity];  
    for (int i = 0; i < size; i++) {  
        new_data[i] = data[i];  
    }  
    data = new_data;  
}
```

ArrayList的实现 - Remove方法

```
public void remove(int index) {  
    if (index < 0 || index >= size) {  
        // throw Exception  
    }  
    size--;  
    for (int i = index; i < size; i++) {  
        data[i] = data[i+1];  
    }  
}
```

ArrayList的实现总结

- 利用数组作为存储
- 初始化时需要指定ArrayList的容量
- 记得边界检查
- 当达到数组容量时再添加新元素时需要resize操作对底层数组进行扩容

总结

- 数组的概念
- 数组典型面试题
- 两根指针算法
- 数组列表的实现