

中山大学网络空间安全学院必修课程

第7节 数组练习

C Programming

岳晟

网络空间安全学院 工学园2栋508

yuesh5@mail.sysu.edu.cn

2026年秋季

有序数组和严格有序数组

- **有序数组**：数组里的元素是**按大小关系顺序排列**的数组
 - 从小到大排列的称为升序数组
 - 当数组中相邻两个元素是严格小于关系时，称为严格升序数组
 - 类似的可以定义降序数组和严格降序数组

升序数组：1, 3, 3, 5, 5, 6

严格升序数组：1, 3, 5, 6

降序数组：6, 5, 3, 3, 3, 1

严格降序数组：6, 5, 3, 1

1. 搜索插入位置

给定一个**严格升序**数组和一个目标值，在数组中找到**目标值**，并返回其**索引**。如果目标值不存在于数组中，返回它将会被按顺序插入的位置。

示例1:

数组: 1, 3, 5, 6

目标值: 5

答案: 2

示例2:

数组: 1, 3, 5, 6

目标值: 2

答案: 1

示例3:

数组: 1, 3, 5, 6

目标值: 7

答案: 4

题目链接: [35. 搜索插入位置 - 力扣 \(Leetcode\)](#)

1. 搜索插入位置

解题方法1： 由于数组是严格升序数组，可以**从左到右依次扫描**数组中的每一个元素，当扫描到的**元素大于等于目标值时**，该位置就是答案位置，后面的元素不需要再扫描；若扫描完整整个数组都没找到满足条件的位置，则输出末尾位置

代码：

```
int search_insert (int a[], int n, int target) {  
    for (int i=0; i<n; i++)  
        if (a[i]>=target) return i;  
    return n;  
}
```

1. 搜索插入位置

解题方法2： 由于数组是严格升序数组，可以使用**二分法**查找答案位置

二分法 (作业)： 通过不断**比较中间元素**，逐步**缩小**查找范围

1. 使用 **l, r** 标记查询范围，初始化 **$l=0, r=n-1$** ，表示 **$[0, n-1]$**
2. 每次取 **$mid=(l+r)/2$** ，比较 $a[mid]$ 与目标值
 - 若 $a[mid] < target$ ，则 **$l=mid+1$**
 - 若 $a[mid] > target$ ，则 **$r=mid-1$**
 - 若 $a[mid] == target$ ，则 **返回 mid**
3. 重复上述过程，直到找到目标或查找范围为空 (**$l > r$**)

1. 搜索插入位置

解题方法2： 由于数组是严格升序数组，可以使用**二分法**查找答案位置

二分法 (本题)： 通过不断**比较中间元素**，逐步**缩小**查找范围

1. 使用 **l, r** 标记查询范围，初始化 **$l=0, r=n-1$** ，表示 **$[0, n-1]$**
2. 每次取 **$mid=(l+r)/2$** ，比较 $a[mid]$ 与目标值
 - 若 $a[mid] < target$ ，则 **$l=mid+1$**
 - 若 $a[mid] > target$ ，则 **$r=mid-1$**
 - 若 $a[mid] == target$ ，则 **返回 mid**
3. 重复，直到找到目标或查找范围为空 (**$l>r$**)；**若未找到，则返回 l**

1. 搜索插入位置

解题方法2： 由于数组是严格升序数组，可以使用**二分法**查找答案位置

二分法 (本题另一实现)： 通过不断**比较中间元素**，逐步**缩小**查找范围

1. 使用 L, R 标记查询范围，初始化 $L = -1, R = n$ 表示 $(-1, n]$
2. 每次判断 $mid = (L + R) / 2$ 元素与目标值的大小关系：若中间元素比目标值小，则 $L = mid$ ；否则 $R = mid$
3. 重复操作，直到 L, R 相邻 (即 $L + 1 == R$)，最终答案就为 R

1. 搜索插入位置

解题方法2： 由于数组是严格升序数组，可以使用**二分法**查找答案位置

二分法 (本题另一实现)： 通过不断**比较中间元素**，逐步**缩小**查找范围

1. 使用 L, R 标记查询范围，初始化 $L = -1, R = n$ 表示 $(-1, n]$
2. 每次判断 $mid = (L + R) / 2$ 元素与目标值的大小关系：若中间元素比目标值小，则 $L = mid$ ；否则 $R = mid$
3. 重复操作，直到 L, R 相邻 (即 $L + 1 == R$)，最终答案就为 R

原因： 数组严格升序，当**中间元素比目标值小时**，**左边元素一定也比目标值小**，故缩小范围到 $(mid, R]$ ；同理，当中间元素大于等于目标值时，右边的元素一定大于目标值，因此缩小范围到 $(L, mid]$

1. 搜索插入位置

假设目标值是 8

第一轮: $L = -1, R = 8, mid = 3; a[mid] = 6 < 8$, 所以 $L = mid = 3$

	<i>L</i>				<i>mid</i>					<i>R</i>
下标	-1	0	1	2	3	4	5	6	7	8
元素	/	1	2	3	6	8	9	10	11	/

第二轮: $L = 3, R = 8, mid = 5; a[mid] = 9 \geq 8$, 所以 $R = mid = 5$

					<i>L</i>		<i>mid</i>			<i>R</i>
下标	-1	0	1	2	3	4	5	6	7	8
元素	/	1	2	3	6	8	9	10	11	/

1. 搜索插入位置

第三轮: $L = 3, R = 5, mid = 4; a[mid] = 8 \geq 8$, 所以 $R = mid = 4$

	L mid R									
下标	-1	0	1	2	3	4	5	6	7	8
元素	/	1	2	3	6	8	9	10	11	/

$L = 3, R = 4$ 满足 $L + 1 == R$, 答案为 $R = 4$

1. 搜索插入位置

```
int binary_search(int a[], int n, int target) {  
    int L=-1, R=n, mid;  
    while (L+1 < R) {  
        mid = (L + R) >> 1; // equals to (L + R) / 2  
        if (a[mid] < target) L = mid;  
        else R = mid;  
    }  
    return R;  
}
```

1. 搜索插入位置

□ 两个解法对比：

- 在最坏情况下，解法1要扫描完整整个数组，即要进行 n 次比较
- 解法2每次可以缩小一半的范围，因此需要进行大约 $\log_2 n$ 次比较
- 当 n 较大时解法2在时间上要优于解法1；但使用解法2的前提是**有序数组**，若是在非有序数组中查找元素，还是需要用解法1的思想

2. 删除有序数组中的重复项

给定一个升序数组，原地**删除数组中重复**出现的元素，使每个元素只出现一次，即去重后的数组为严格升序数组，并返回去重后数组的长度。

示例：

数组：0, 0, 1, 1, 1, 2, 2, 3, 3, 4

去重后数组：0, 1, 2, 3, 4

长度：5

题目链接：[26. 删除有序数组中的重复项 - 力扣 \(Leetcode\)](#)

2. 删除有序数组中的重复项

- ❑ 直接想法是新建一个数组，从左到右扫描原数组的元素，如果当前元素与新数组的末尾元素不一样，则把当前元素添加到新数组中，否则不添加
- ❑ 试着不新建数组，将去重后的数组放在原数组的前部：
 1. 记原数组为 a ，用变量 m 表示去重后数组下一个元素的位置，可知 m 也是去重数组大小
 2. 从左到右扫描原数组（下标记为 i ），如果 $a[m - 1] \neq a[i]$ ，则 $a[m] = a[i]$ ， $m++$ ，否则不进行操作
 3. 由于 $m \leq i$ 始终成立，不会出现未扫描的位置被覆盖的情况

2. 删除有序数组中的重复项

初始: $i = 1, m = 1$ (非空数组第 0 位一定保留)

				i
1	2	3	3	3
			m	

$a[m - 1] == a[i]$, 不保留 $a[i]$

	i			
1	1	2	3	3
		m		

$i = 2, m = 1, a[m - 1] != a[i]$, 保留 $a[i]$

	i			
1	1	2	3	3
		m		

$i = 3, m = 2, a[m - 1] != a[i]$, 保留 $a[i]$

		i		
1	2	2	3	3
		m		

结束, $m = 3$

```
int removeDuplicates(int a[], int n) {  
    if (n==0) return 0;  
    int m=1;  
    for (int i=1; i<n; ++i)  
        if (a[i]!=a[m-1]) a[m++]=a[i];  
    return m;  
}
```

3. 两数之和

给定一个数组与一个目标值，找到数组中的**两个不同位置**，满足位置上的元素**相加等于目标值**。数据保证有唯一解。

示例1：

数组：2, 7, 11, 15

目标值：9

答案（下标）：0, 1

示例2：

数组：3, 2, 4

目标值：6

答案：1, 2

示例3：

数组：3, 3

目标值：6

答案：0, 1

题目链接：[1. 两数之和 - 力扣 \(Leetcode\)](#)

3. 两数之和

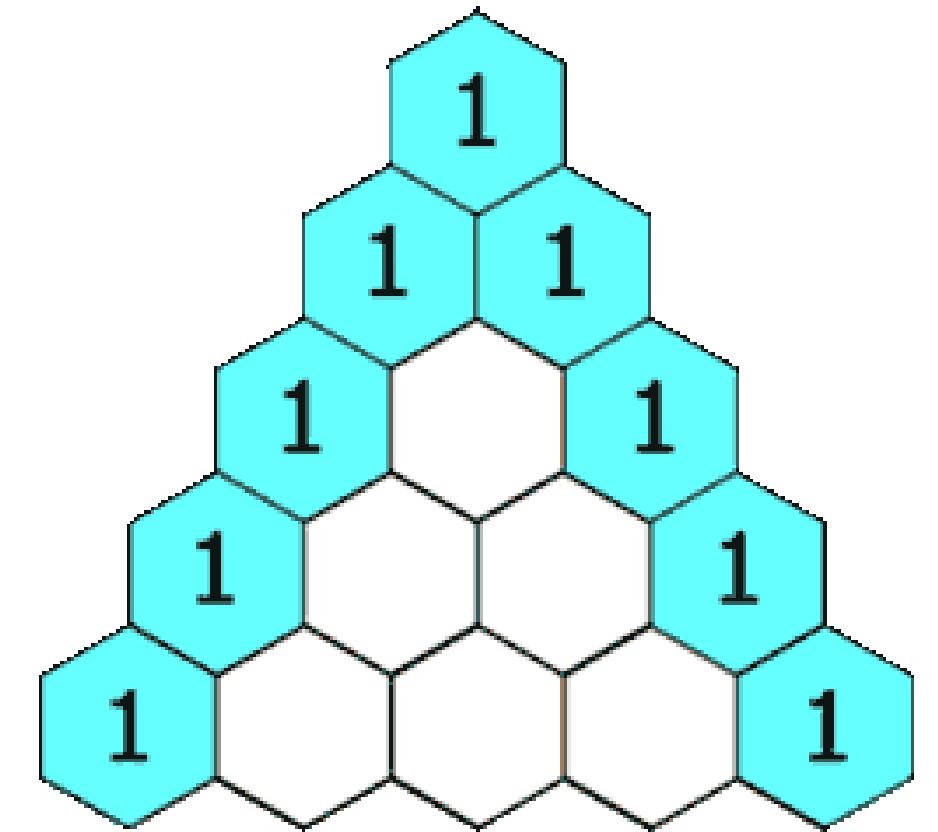
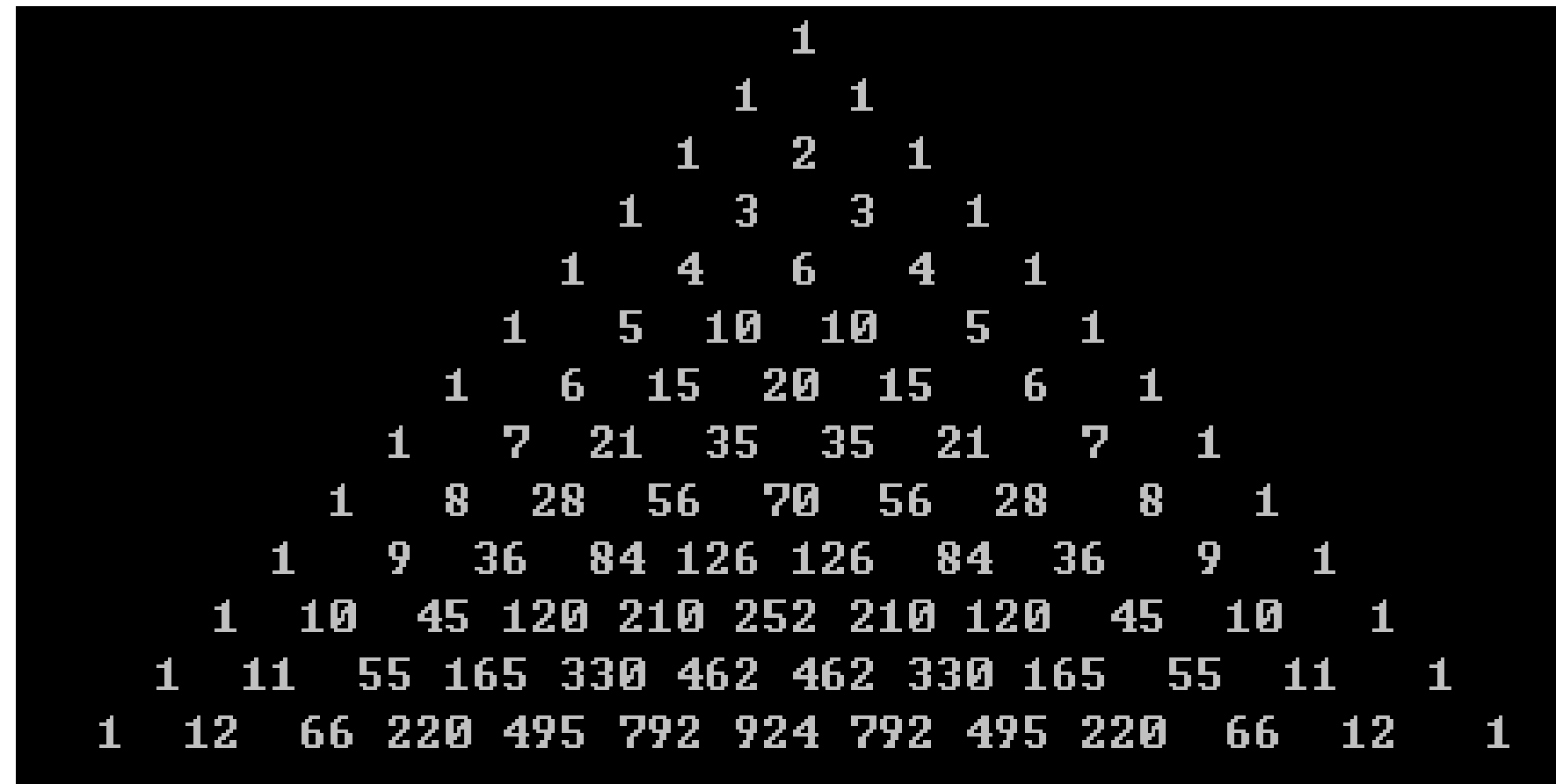
解法：枚举所有的下标组合，判断哪一组对应元素相加等于目标值

代码：

```
void twoSum(int a[], int n, int target, int idx[]) {  
    for (int i=0; i<n-1; ++i)  
        for (int j=i+1; j<n; ++j)  
            if (a[i]+a[j]==target) {  
                idx[0]=i;  
                idx[1]=j;  
                return;  
            }  
}
```

4. 打印杨辉三角形

杨辉三角如图：



- ❑ 输入： $n \geq 1 \ \&\& \ n \leq 13$
- ❑ 解法一：可以使用二维数组
- ❑ 解法二：要求使用一个最多 n 个元素的数组

题目链接：[119. 杨辉三角 II - 力扣 \(Leetcode\)](https://leetcode.cn/problems/pascals-triangle-ii/)

4. 打印杨辉三角形

```
void print_yanghui(int n){  
    int line[n];  
    for (int i=1; i<=n; i++) {  
        current_line(line, i);  
        for (int j=0; j<i; j++)  
            printf("%-4d", line[j]);  
        printf("\n");  
    }  
}
```

4. 打印杨辉三角形

第4行（原数组）

0	1	2	3	4
1	3	3	1	—

↓ 相邻两个元素相加，首尾为1

第5行（目标结果）

0	1	2	3	4
1	4	6	4	1

4. 打印杨辉三角形

解法：

首先，第1、2行是基本情况

其次，利用计算 i 位置值依赖 i 和 $i-1$ 位置数值这个特点，可以在原数组上**从右向左**计算

```
void current_line(int a[], int line_no) {  
    if (line_no == 1) a[0] = 1;  
    else if (line_no == 2) a[0] = a[1] = 1;  
    else {  
        // 从右向左更新中间元素  
        for (int i=line_no-2; i>=1; i--) {  
            a[i] = a[i-1] + a[i];  
        }  
        // 设置首尾元素  
        a[0] = 1;  
        a[line_no-1] = 1;  
    }  
}
```

4. 打印杨辉三角形

解法：

首先，第1、2行是基本情况

其次，利用计算 i 位置值依赖 i 和 $i-1$ 位置数值这个特点，可以在原数组上**从右向左**计算

```
void current_line(int a[], int line_no) {  
    if (line_no == 1) a[0] = 1;  
    else if (line_no == 2) a[0] = a[1] = 1;  
    else {  
        // 从右向左更新中间元素  
        for (int i=line_no-2; i>=1; i--) {  
            a[i] = a[i-1] + a[i];  
        }  
        // 设置首尾元素  
        a[0] = 1;  
        a[line_no-1] = 1;  
    }  
}
```

案例与知识点总结

□ 搜索插入位置

- 单指针方法：使用 i 顺序扫描
- 二分查找方法：双指针方法一种，每次根据中间 m 位置的值，将 $(L,R]$ 区间，在 $(L,m]$ 、 $(m,R]$ 区间中取其一（该方法仅适用有序数组）

□ 删除有序数组中的重复项

- 双指针方法：运用前、后指针特点，实现在一个数组上完成重复项删除

□ 两数之和

- 暴力法方法：通过枚举所有组合，虽然简单粗暴，也很优美

□ 打印杨辉三角形

- 前后数据依赖关系分析，例如：后一行依赖前一行等

谢谢