普通数组用法

```cpp
#include <iostream>

int main() {
    // 1. Initialize
    int a0[5];
    int a1[5] = {1, 2, 3};  // other element will be set as the default value
    // 2. Get Length
    int size = sizeof(a1) / sizeof(*a1);
    cout << "The size of a1 is: " << size << endl;
    // 3. Access Element
    cout << "The first element is: " << a1[0] << endl;
    // 4. Iterate all Elements
    cout << "[Version 1] The contents of a1 are:";
    for (int i = 0; i < size; ++i) {
        cout << " " << a1[i];
    }
    cout << endl;
    cout << "[Version 2] The contents of a1 are:";
    for (int& item: a1) {
        cout << " " << item;
    }
    cout << endl;
    // 5. Modify Element
    a1[0] = 4;
    // 6. Sort
    sort(a1, a1 + size);
```

动态数组用法

```cpp
#include <iostream>

int main() {
    // 1. initialize
    vector<int> v0;
    vector<int> v1(5, 0);
    // 2. make a copy
    vector<int> v2(v1.begin(), v1.end());
    vector<int> v3(v2);
    // 2. cast an array to a vector
    int a[5] = {0, 1, 2, 3, 4};
    vector<int> v4(a, *(&a + 1));
    // 3. get length
    cout << "The size of v4 is: " << v4.size() << endl;
    // 4. access element
    cout << "The first element in v4 is: " << v4[0] << endl;
    // 5. iterate the vector
```

```
    cout << "[Version 1] The contents of v4 are:";
    for (int i = 0; i < v4.size(); ++i) {
        cout << " " << v4[i];
    }
    cout << endl;
    cout << "[Version 2] The contents of v4 are:";
    for (int& item : v4) {
        cout << " " << item;
    }
    cout << endl;
    cout << "[Version 3] The contents of v4 are:";
    for (auto item = v4.begin(); item != v4.end(); ++item) {
        cout << " " << *item;
    }
    cout << endl;
    // 6. modify element
    v4[0] = 5;
    // 7. sort
    sort(v4.begin(), v4.end());
    // 8. add new element at the end of the vector
    v4.push_back(-1);
    // 9. delete the last element
    v4.pop_back();
}
```

724.寻找数组的中心索引（简单）

1. 题目描述

给定一个整数类型的数组 nums，请编写一个能够返回数组“中心索引”的方法。

我们是这样定义数组中心索引的：数组中心索引的左侧所有元素相加的和等于右侧所有元素相加的和。如果数组不存在中心索引，那么我们应该返回 -1。如果数组有多个中心索引，那么我们应该返回最靠近左边的那一个。

示例 1:

```
输入:
nums = [1, 7, 3, 6, 5, 6]
输出: 3
解释:
索引3 (nums[3] = 6) 的左侧数之和(1 + 7 + 3 = 11)，与右侧数之和(5 + 6 = 11)相等。
同时，3 也是第一个符合要求的中心索引。
```

示例 2:

```
输入:
nums = [1, 2, 3]
输出: -1
解释:
数组中不存在满足此条件的中心索引。
```

说明:

- nums 的长度范围为 [0, 10000]。
- 任何一个 nums[i] 将会是一个范围在 [-1000, 1000]的整数。

2. 简单实现

用sum[i]统计nums[0]+...+nums[i]即可

```
class Solution {
public:
    int pivotIndex(vector<int>& nums) {
        int len = nums.size();
        if(len == 0)
            return -1;
        else if(len == 1)
            return 0;
        //统计sum
        vector<int> sum = vector<int>(len, 0);
        sum[0] = nums[0];
        for(int i = 1; i < len; i++){
            sum[i] = sum[i-1] + nums[i];
        }
        //查找中心
        for(int i = 0; i < len; i++){
            if(sum[i]-nums[i] == sum[len-1]-sum[i])
                return i;
        }
        return -1;
    }
};
```

747.至少是其他数字两倍的最大数（简单）

1. 题目描述

在一个给定的数组nums中，总是存在一个最大元素.查找数组中的最大元素是否至少是数组中每个其他数字的两倍。如果是，则返回最大元素的索引，否则返回-1。

示例 1:

```
输入: nums = [3, 6, 1, 0]
输出: 1
解释: 6是最大的整数，对于数组中的其他整数，
6大于数组中其他元素的两倍。6的索引是1，所以我们返回1.
```

示例 2:

```
输入: nums = [1, 2, 3, 4]
输出: -1
解释: 4没有超过3的两倍大，所以我们返回 -1.
```

提示:

- nums 的长度范围在[1, 50].
- 每个 nums[i] 的整数范围在 [0, 100].

2. 简单实现

- 只要最大数比次大数的两倍大，就一定比其他数的两倍大
- 题目描述对于存在相同数的处理规则不明确，通过自定义测试用例可以看到系统的判断规则

```
class Solution {
public:
    int dominantIndex(vector<int>& nums) {
        int len = nums.size();
        if(len == 1)
            return 0;
        int max1;
        int max2 = -1;
        int idx;
        if(nums[0] > nums[1]){
            max1 = nums[0];
            max2 = nums[1];
            idx = 0;
        }
        else if(nums[0] == nums[1]){
            max1 = nums[0];
            idx = 0;
        }
        else {
            max1 = nums[1];
            max2 = nums[0];
            idx = 1;
        }
        for(int i = 2; i < len; i++){
            if(nums[i] > max1){//(max1, +)
                max2 = max1;
                max1 = nums[i];
                idx = i;
            }
            else if(nums[i] < max1 && nums[i] > max2)//(max2,max1)
                max2 = nums[i];
        }
        if(max1 >= 2*max2)
            return idx;
        else
            return -1;
    }
};
```

66.加一（简单）

1. 题目描述

给定一个由整数组成的非空数组所表示的非负整数，在该数的基础上加一。最高位数字存放在数组的首位， 数组中每个元素只存储单个数字。你可以假设除了整数 0 之外，这个整数不会以零开头。

示例 1:

```
输入: [1,2,3]
输出: [1,2,4]
解释: 输入数组表示数字 123。
```

示例 2:

```
输入: [4,3,2,1]
输出: [4,3,2,2]
解释: 输入数组表示数字 4321。
```

2. 简单实现

```
class Solution {
public:
    vector<int> plusOne(vector<int>& digits) {
        int i = digits.size() - 1;
        for(; i >= 0; i--){
            if(digits[i] <= 8){
                digits[i] += 1;
                break;
            }
            else
                digits[i] = 0;
        }
        if(digits[0] == 0)
            digits.insert(digits.begin(), 1);
        return digits;
    }
};
```

498.对角线遍历（中等）

1. 题目描述

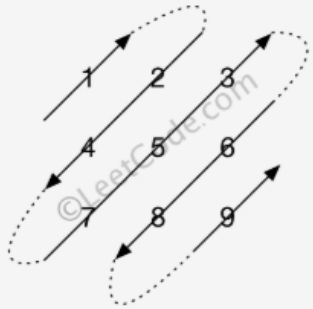
给定一个含有 M x N 个元素的矩阵（M 行，N 列），请以对角线遍历的顺序返回这个矩阵中的所有元素，对角线遍历如下图所示。

示例:

输入:
```
[
 [ 1, 2, 3 ],
 [ 4, 5, 6 ],
 [ 7, 8, 9 ]
]
```

输出: `[1,2,4,7,5,3,6,8,9]`

解释:



说明:

1. 给定矩阵中的元素总数不会超过 100000 。

2. 简单实现

```
class Solution {
public:
    vector<int> findDiagonalOrder(vector<vector<int>>& matrix) {
        vector<int> ans;
        int m = matrix.size();
        if(m == 0)
            return ans;
        int n = matrix[0].size();

        int dire[2] = {-1, 1};
        int x = 0, y = 0;
        while(1){
            ans.push_back(matrix[x][y]);
            if(x == m-1 && y == n-1)
                break;
            if(dire[0] < 0){//右上
                if(y == n-1){
                    x++;
                    dire[0] = -dire[0];
                    dire[1] = -dire[1];
                }
                else if(x == 0){
```

```
                    y++;
                    dire[0] = -dire[0];
                    dire[1] = -dire[1];
                }
                else{
                x += dire[0];
                y += dire[1];
                }
            }
            else{//左下
                if(x == m-1){
                    y++;
                    dire[0] = -dire[0];
                    dire[1] = -dire[1];
                }
                else if(y == 0){
                    x++;
                    dire[0] = -dire[0];
                    dire[1] = -dire[1];
                }
                else{
                    x += dire[0];
                    y += dire[1];
                }
            }
        }
        return ans;
    }
};
```

54.螺旋矩阵（中等）

1. 题目描述

   给定一个包含 *m* x *n* 个元素的矩阵（*m* 行, *n* 列），请按照顺时针螺旋顺序，返回矩阵中的所有元素。

   示例 1:

```
输入:
[
 [ 1, 2, 3 ],
 [ 4, 5, 6 ],
 [ 7, 8, 9 ]
]
输出: [1,2,3,6,9,8,7,4,5]
```

   示例 2:

```
输入:
[
  [1, 2, 3, 4],
  [5, 6, 7, 8],
  [9,10,11,12]
]
输出: [1,2,3,4,8,12,11,10,9,5,6,7]
```

2. 简单实现

```
class Solution {
public:
    vector<int> spiralOrder(vector<vector<int>>& matrix) {
        vector<int> ans;
        int m = matrix.size();
        if(m <= 0)
            return ans;
        int n = matrix[0].size();

        int pad = 1;
        while(2*pad <= m && 2*pad <= n){
            int i = pad-1;
            int j = pad-1;
            while(j < n-pad+1){
                ans.push_back(matrix[i][j]);
                j++;
            }
            j--;
            i++;
            while(i < m-pad+1){
                ans.push_back(matrix[i][j]);
                i++;
            }
            i--;
            j--;
            while(j >= pad-1){
                ans.push_back(matrix[i][j]);
                j--;
            }
            j++;
            i--;
            while(i > pad-1){
                ans.push_back(matrix[i][j]);
                i--;
            }
            pad++;
        }
        int temp = 2*(pad-1);
        if(m == n){
            if(m % 2)
                ans.push_back(matrix[m/2][n/2]);
            return ans;
```

```
        }
        else if(m > n && n%2){
            int j = pad-1;
            for(int i = pad-1; i < m-pad+1; i++)
                ans.push_back(matrix[i][j]);
        }
        else if(m < n && m%2){
            int i = pad-1;
            for(int j = pad-1; j < n-pad+1; j++)
                ans.push_back(matrix[i][j]);
        }
        return ans;
    }
};
```

字符串

29.实现strStr()（简单

1. 题目描述

   实现 strStr() 函数。

   给定一个 haystack 字符串和一个 needle 字符串，在 haystack 字符串中找出 needle 字符串出现的第一个位置 (从0开始)。如果不存在，则返回 **-1**。

   示例 1:

   ```
   输入: haystack = "hello", needle = "ll"
   输出: 2
   ```

   示例 2:

   ```
   输入: haystack = "aaaaa", needle = "bba"
   输出: -1
   ```

   说明:

   - 当 `needle` 是空字符串时，我们应当返回什么值呢？这是一个在面试中很好的问题。
   - 对于本题而言，当 `needle` 是空字符串时我们应当返回 0 。这与C语言的 strstr() 以及 Java的 indexOf() 定义相符。

2. 简单实现

   KMP算法

```
class Solution {
public:
    void get_next(string s, vector<int> &next){
        next[0] = 0;
        int j = 0;//重复前缀
        int i = 1;//当前位置
```

```
        while(i < s.size()){
            if(s[i] == s[j]){
                next[i] = j + 1;
                i++;
                j++;
            }
            else{
                if(j == 0){//无重复前缀
                    next[i] = 0;
                    i++;
                }
                else
                    j = next[j-1];//寻找更短的重复前缀
            }
        }
    }
    int strStr(string haystack, string needle) {
        if(needle == "")
            return 0;
        vector<int> next = vector<int>(needle.size(), 0);
        get_next(needle, next);
        int i = 0;
        int j = 0;
        while(i < haystack.size() && j < needle.size()){
            if(haystack[i] == needle[j]){
                i++;
                j++;
            }
            else{
                if(j != 0)
                    j = next[j-1];
                else
                    i++;
            }
        }
        if(j == needle.size())
            return i - needle.size();
        else
            return -1;
    }
};
```

14.最长公共前缀（简单）

1. 题目描述

   编写一个函数来查找字符串数组中的最长公共前缀。

   如果不存在公共前缀，返回空字符串 `""`。

   示例 1:

```
输入: ["flower","flow","flight"]
输出: "fl"
```

示例 2:

```
输入: ["dog","racecar","car"]
输出: ""
解释: 输入不存在公共前缀。
```

说明:

- 所有输入只包含小写字母 `a-z` 。

2. 简单实现

```
class Solution {
public:
    string longestCommonPrefix(vector<string>& strs) {
        string ans = "";
        int size = strs.size();
        if(size == 0)
            return ans;
        else if(size == 1)
            return strs[0];
        int min_len = strs[0].size();
        for(int i = 1; i < size; i++)
            if(strs[i].size() < min_len)
                min_len = strs[i].size();

        for(int idx = 0; idx < min_len; idx++){
            char temp = strs[0][idx];
            for(int i = 1; i < size; i++){
                if(strs[i][idx] != temp)
                    return ans;
            }
            ans += temp;
        }
       return ans;
    }
};
```

双指针

561.数组拆分 I（简单）

1. 题目描述

给定长度为 **2n** 的数组, 你的任务是将这些数分成 **n** 对, 例如 (a1, b1), (a2, b2), ..., (an, bn) ， 使得从1 到 n 的 min(ai, bi) 总和最大。

示例 1:

```
输入: [1,4,3,2]

输出: 4
解释: n 等于 2, 最大总和为 4 = min(1, 2) + min(3, 4).
```

提示:

1. **n** 是正整数,范围在 [1, 10000].
2. 数组中的元素范围在 [-10000, 10000].

2. 简单实现

先排序，再取奇数位元素值的和

```
class Solution {
public:
    int partition(int l, int r, vector<int>& nums){
        if(l >= r)
            return r;
        int aim = nums[l];
        int i = l + 1, j = r;
        while(i <= j){

            while(i <= j && nums[i] <= aim)
                i++;
            while(i <= j && nums[j] >= aim)
                j--;
            if(i < j)
                swap(nums[i], nums[j]);
        }
        if(l != j)
            swap(nums[l], nums[j]);
        return j;
    }
    void quicksort(int l, int r, vector<int>& nums){
        if(l>=r) return;
        int p = partition(l, r, nums);
        quicksort(l, p-1, nums);
        quicksort(p+1, r, nums);
    }
    int arrayPairSum(vector<int>& nums) {
        quicksort(0, nums.size()-1, nums);
        int size = nums.size()/2;
        int ans = 0;
        for(int i = 0; i < size; i++)
            ans += nums[i*2];
        return ans;
    }
};
```

485.最大连续1的个数（简单）

1. 题目描述

   给定一个二进制数组， 计算其中最大连续1的个数。

   示例 1:

   ```
   输入: [1,1,0,1,1,1]
   输出: 3
   解释: 开头的两位和最后的三位都是连续1，所以最大连续1的个数是 3.
   ```

   注意:

   - 输入的数组只包含 `0` 和 `1`。
   - 输入数组的长度是正整数，且不超过 10,000。

2. 简单实现

```cpp
class Solution {
public:
    int findMaxConsecutiveOnes(vector<int>& nums) {
        int max = 0;
        int temp = 0;
        int len = nums.size();
        for(int i = 0; i < len; i++){
            if(nums[i] == 1)
                temp ++;
            else if(temp > 0){
                if(temp > max)
                    max = temp;
                temp = 0;
            }
            if(temp + len - i -1 <= max)//即使剩下的全是1也不可能长度最大了
                return max;
        }
        return temp;
    }
};
```

209.长度最小的子数组（中等）

1. 题目描述

   给定一个含有 **n** 个正整数的数组和一个正整数 **s** ，找出该数组中满足其和 **≥ s** 的长度最小的连续子数组。如果不存在符合条件的连续子数组，返回 0。

   示例:

   ```
   输入: s = 7, nums = [2,3,1,2,4,3]
   输出: 2
   解释: 子数组 [4,3] 是该条件下的长度最小的连续子数组。
   ```

   进阶:

   - 如果你已经完成了*O*(*n*) 时间复杂度的解法, 请尝试 *O*(*n* log *n*) 时间复杂度的解法。

2. 简单实现

   双指针+滑动窗口

```
class Solution {
public:
    int minSubArrayLen(int s, vector<int>& nums){
        int n = nums.size();
        int ans = INT_MAX;//最短的目标连续子数组长度
        int left = 0;//窗口最左端
        int sum = 0;//当前窗口和
        for (int i = 0; i < n; i++) {
            sum += nums[i];
            while (sum >= s) {
                ans = min(ans, i + 1 - left);//更新ans
                sum -= nums[left++];//窗口左端右移
            }
        }
        return (ans != INT_MAX) ? ans : 0;
    }
};
```

189.旋转数组（简单）

1. 题目描述

   给定一个数组，将数组中的元素向右移动 *k* 个位置，其中 *k* 是非负数。

   示例 1:

```
输入: [1,2,3,4,5,6,7] 和 k = 3
输出: [5,6,7,1,2,3,4]
解释:
向右旋转 1 步: [7,1,2,3,4,5,6]
向右旋转 2 步: [6,7,1,2,3,4,5]
向右旋转 3 步: [5,6,7,1,2,3,4]
```

   示例 2:

```
输入: [-1,-100,3,99] 和 k = 2
输出: [3,99,-1,-100]
解释:
向右旋转 1 步: [99,-1,-100,3]
向右旋转 2 步: [3,99,-1,-100]
```

   说明:

   - 尽可能想出更多的解决方案，至少有三种不同的方法可以解决这个问题。
   - 要求使用空间复杂度为 O(1) 的 **原地** 算法。

2. 简单实现

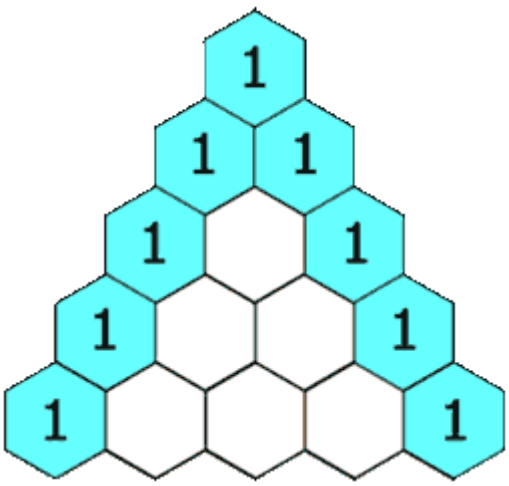
   三次inverse

```
class Solution {
public:
    void inverse(vector<int>& nums, int i, int j)
    {
        int temp;
        while(i<j)
        {
            temp=nums[i];
            nums[i]=nums[j];
            nums[j]=temp;
            i++;
            j--;
        }
    }
    void rotate(vector<int>& nums, int k) {
        int the_size = nums.size();
        k=k%the_size;
        inverse(nums,0,the_size-k-1);
        inverse(nums,the_size-k,the_size-1);
        inverse(nums,0,the_size-1);
    }
};
```

119.杨辉三角 II(简单)

1. 题目描述

给定一个非负索引 *k*，其中 *k* ≤ 33，返回杨辉三角的第 *k* 行。



在杨辉三角中，每个数是它左上方和右上方的数的和。

示例:

```
输入: 3
输出: [1,3,3,1]
```

进阶:

- 你可以优化你的算法到 *O*(*k*) 空间复杂度吗?

2. 简单实现

用pre代表上一行，求当前行cur

```
class Solution {
public:
    vector<int> getRow(int rowIndex) {
        if(rowIndex == 0)
            return vector<int>(1,1);
        else if(rowIndex == 1)
            return vector<int>(2,1);

        vector<int> pre = vector<int>(2,1);
        vector<int> cur = vector<int>(pre.size()+1, 1);
        int idx = 2;
        while(idx <= rowIndex){
            for(int i = 1; i < cur.size()-1; i++){
                cur[i] = pre[i-1] + pre[i];
            }
            pre = cur;
            cur = vector<int>(pre.size()+1, 1);
            idx++;
        }
        return pre;
    }
};
```

3. 最佳算法 杨辉三角第k行为$(x+y)^2$的系数，第k行第i个系数为C_k^i

```
class Solution {
public:
    vector<int> getRow(const int rowIndex) {
        vector<int> ans;
        for (int i = 0; i <= rowIndex; ++i)
            ans.push_back(combination(rowIndex, i));
        return ans;
    }

private:
    int combination(const int m, const int i) {//计算C(m,i)
        int n = min(i, m - i); // C(m,i) = C(m,m-i)
        int k = m - n;
        long long c = 1;
        for (int j = 1; j <= n; ++j) {
            c *= (k + j);
            c /= j;
        }
        return c;
    }
};
```

Tips：combination函数有一些细节值得注意：

- c因为连乘数值大，要用long long
- j要从小到大，否则可能无法整除
- 不能c *= (k+j)/j，因为k+j不一定能整除j

151.翻转字符串里的单词（中等）

1. 题目描述

   给定一个字符串，逐个翻转字符串中的每个单词。

   示例 1：

   ```
   输入: "the sky is blue"
   输出: "blue is sky the"
   ```

   示例 2：

   ```
   输入: "  hello world!  "
   输出: "world! hello"
   解释: 输入字符串可以在前面或者后面包含多余的空格，但是反转后的字符不能包括。
   ```

   示例 3：

   ```
   输入: "a good   example"
   输出: "example good a"
   解释: 如果两个单词间有多余的空格，将反转后单词间的空格减少到只含一个。
   ```

   说明：

   - 无空格字符构成一个单词。
   - 输入字符串可以在前面或者后面包含多余的空格，但是反转后的字符不能包括。
   - 如果两个单词间有多余的空格，将反转后单词间的空格减少到只含一个。

   进阶： 请选用 C 语言的用户尝试使用 *O*(1) 额外空间复杂度的原地解法。

2. 简单实现

   空格分词然后倒序加到ans即可

   ```
   class Solution {
   public:
       string reverseWords(string s) {
           string ans = "";
           string temp = "";
           for(int i = 0; i < s.size(); i++){
               if(s[i] == ' '){
                   if(temp != ""){
                       if(ans != "")
                           ans = temp + ' ' + ans;
                       else
                           ans = temp;
   ```

```
                    temp = "";
                }
            }
            else
                temp += s[i];
        }
        if(temp != "")
            if(ans != "")
                ans = temp + ' ' + ans;
            else
                ans = temp;
        return ans;
    }
};
```

3. Tips: string再后面append比在前面插入要省时间，所以可以用栈

```
class Solution {
public:
    string reverseWords(string s) {
        stack<string> st;
        string temp = "";
        for(int i = 0; i < s.size(); i++){
            if(s[i] == ' '){
                if(temp != ""){
                    st.push(temp);
                    temp = "";
                }
            }
            else{
                temp += s[i];
            }
        }
        if(temp != "")
            st.push(temp);
        string ans = "";
        if(st.empty())
            return ans;
        else{
            ans = st.top();
            st.pop();
            while(!st.empty()){
                ans += ' ' + st.top();
                st.pop();
            }
        }
        return ans;
    }
};
```

今后也要避免在string前面加入字符串

4. 原地算法

   先整体翻转，再逐单词反转

```
class Solution {
public:
    string reverseWords(string s) {
        reverse(s.begin(), s.end());
        string ans = "";
        string temp = "";
        for(int i = 0; i < s.size(); i++){
            if(s[i] == ' '){
                if(temp != ""){
                    reverse(temp.begin(), temp.end());
                    if(ans == "")
                        ans += temp;
                    else
                        ans += ' ' + temp;
                    temp = "";
                }
            }
            else
                temp += s[i];
        }
        if(temp != ""){
            reverse(temp.begin(), temp.end());
            if(ans == "")
                ans += temp;
            else
                ans += ' ' + temp;
        }
        return ans;
    }
};
```

557.反转字符串中的单词 III（简单）

1. 题目描述

   给定一个字符串，你需要反转字符串中每个单词的字符顺序，同时仍保留空格和单词的初始顺序。

   示例 1:

```
输入: "Let's take LeetCode contest"
输出: "s'teL ekat edoCteeL tsetnoc"
```

   注意：在字符串中，每个单词由单个空格分隔，并且字符串中不会有任何额外的空格。

2. 简单实现

   以空格为标识逐个反转单词即可

```
class Solution {
```

```
public:
    string reverseWords(string s) {
        int last = 0;
        int i = 0;
        for(i = 0; i < s.length(); i++){
            if(s[i] == ' '){
                reverse(s.begin()+last, s.begin()+i);
                last = i + 1;
            }
        }
        reverse(s.begin()+last, s.begin()+i);
        return s;
    }
};
```

283.移动零（简单）

1. 题目描述

   给定一个数组 `nums`，编写一个函数将所有 `0` 移动到数组的末尾，同时保持非零元素的相对顺序。

   示例:

   ```
   输入: [0,1,0,3,12]
   输出: [1,3,12,0,0]
   ```

   说明:

   1. 必须在原数组上操作，不能拷贝额外的数组。
   2. 尽量减少操作次数。

2. 简单实现

```
class Solution {
public:
    void moveZeroes(vector<int>& nums) {
        int i = 0;//当前数组第一个0的idx
        int j = 0;//当前数组第i个元素之后第一个非0值的idx
        while(j < nums.size()){
            while(i < nums.size() && nums[i] != 0) i++;
            while(j < nums.size() && nums[j] == 0) j++;
            if(i < j && j < nums.size())
                swap(nums[i], nums[j]);
            else
                while(j < nums.size() && nums[j] != 0) j++;//找到nums[i]之后的第一个非0值
        }
       return;
    }
};
```