

数组算法框架

1. 矩阵转置 矩阵转置，即将矩阵的上三角和下三角元素互换。对角线元素不变。 框架；

```
// row == col 的情况
for(int i=0;i<row;i++) {
    for(int j=i+1;j<col;j++) {
        std::swap(nums[i][j],nums[j][i]);
    }
}

// 普通矩阵
std::vector<std::vector<int>> nums(col, std::vector<int>(row));
for(int i=0;i<row;i++) {
    for(int j=0;j<col;j++){
        nums[j][i] = matrix[i][j];
    }
}
```

2. 数组旋转 90 度

分析：将数组转置后，每行逆置

```
// 普通矩阵
std::vector<std::vector<int>> nums(col, std::vector<int>(row));
for(int i=0;i<row;i++) {
    for(int j=0;j<col;j++){
        nums[j][i] = matrix[i][j];
    }
}

for(int i=0;i<col;i++) {
    int left = 0, right = row-1;
    while(left < right) {
        std::swap(nums[i][left++], nums[i][right--]);
    }
}
```

3. 旋转打印数组

```
class Solution {
public:
    std::vector<int> spiralOrder(std::vector<std::vector<int>>& matrix) {
        std::vector<int> ans;

        int row = matrix.size(), col = matrix[0].size();
```

```

int beginx = 0; // 标记最左侧列的列号，也表示第一行开始
int endx = col - 1; // 标记最右侧列的列号，也表示第一行结束，
int beginy = 0; // 标记最上面一行的行号
int endy = row - 1; // 标记最下面一行的行号

while (true) {
    // 遍历第一行
    for (int i = beginx; i <= endx; i++) {
        ans.push_back(matrix[beginy][i]);
    }
    beginy++; // 下一行
    if (beginy > endy) {
        break;
    }

    // 遍历最后一列
    for (int i = beginy; i <= endy; i++) {
        ans.push_back(matrix[i][endx]);
    }
    endx--;
    if (beginx > endx) {
        break;
    }

    // 最后一行
    for (int i = endx; i >= beginx; i--) {
        ans.push_back(matrix[endy][i]);
    }
    endy--;
    if (beginy > endy) {
        break;
    }

    // 第一列
    for (int i = endy; i >= beginy; i--) {
        ans.push_back(matrix[i][beginx]);
    }
    beginx++;
    if (beginx > endx) {
        break;
    }
}

return ans;
}
};

```

4, 流式分割字符串

```

class Solution {
public:
    int countSegments(std::string s) {

```

```

int            ans    = 0;
std::string    split = " ";
std::stringstream ss(s);

while (ss >> split) {
    ans++;
}

return ans;
}
};

```

5. 求中位数算法 a. 两个长度相等的排序数组中求上中位数 分析：

- 长度相等的排序数组，相对较为容易，可以通过查找两个数组的中位数来判断求上中位数；
 - 如果两个数组的 mid 相同，则无论数组长度的奇偶性，该值一定为其上中位数。
 - 如果两个数组的 mid 部分中位数不同，
 - 当 `nums1[mid1] > nums2[mid2]` 时，则中位数一定在 `nums1` 的 `[mid1+offset, end1]` 与 `[start2, mid2]` 的区间中；
 - 当 `nums1[mid1] < nums2[mid2]` 时，则中位数一定在 `nums1` 的 `[start1, mid1]` 与 `[mid2+offset, end2]` 的区间中；
 - `offset` 为偏移量，长度为偶数值为 1，长度为奇数值为 0。
 - 当 `start1 == end1` 时，仍未找相同 mid 位置，则表示其上中位数为 `nums1[start1]`，`nums2[start2]` 的最小值。

```

// 两个长度相等的排序数组中求上中位数
int getUpMedian(std::vector<int> &nums, std::vector<int> &vec) {
    if (nums.empty() || vec.empty()) {
        return -1;
    }
    int start1 = 0, end1 = nums.size() - 1;
    int start2 = 0, end2 = vec.size() - 1;
    int mid1 = 0, mid2 = 0;
    int offset = 0;
    while (start1 < end1) {
        mid1 = start1 + (end1 - start1) / 2;
        mid2 = start2 + (end2 - start2) / 2;
        // offset标记end的便宜位置，偶数为1，奇数为0
        offset = ((end1 - start1 + 1) & 1) ^ 1;

        if (nums[mid1] < vec[mid2]) {
            start1 = mid1 + offset;
            end2 = mid2;
        } else if (nums[mid1] > vec[mid2]) {
            start2 = mid2 + offset;
            end1 = mid1;
        } else {
            return nums[mid1];
        }
    }
}

```

```
return std::min(nums[start1], vec[start2]);
}
```

6. 求从1~n中1的个数 可以利用以下公式：

(最高位数字) * (除去最高位后剩余的位数) * (某一位固定为1时余下的从0~9的自由变化数)

```
int solution(int num){
    if(num < 1) {
        return 0;
    }

    int len = getLenOfNum(num); // 计算数据位数
    if(len == 1) {
        return 1;
    }
    int tmp = powerBaseOf10(len - 1); // 计算给定数据最高位代表的分位值
    int first = num / tmp; // 计算当前数字的首位数字
    int firstOneNum = first == 1 ? num % tmp + 1 : tmp; // 当首位数字为1时，出现的1的次数仅为：当前数字与最高分位值余数+1，否则一定为当前最高分位值个数
    int otherOneNum = first * (len - 1) * (tmp / 10); // 固定首位之后，其余部分值为tmp / 10
    return firstOneNum + otherOneNum + solution(num % tmp);
}

int getLenOfNum(int num) {
    int len = 0;

    while(len != 0){
        len++;
        num /= 10;
    }

    return len;
}

int powerBaseOf10(int base) {
    return power(10, base);
}
```

7. 求一个正整数数组中未出现的最小整数

分析：定义区间[1...l]表示该部分元素已经完成排序，[l+1...r]表示需要进行排序的元素。那么必定存在：r表示这个边界，如果大于这个r的数就会被扔掉。r的初值为N表示[1-r]的元素都不会被扔掉，大于r的就会被扔掉。但是这个r的值是变化的，如果[l+1~r]中有一个元素不合法，那么这个r就是减少1，因为最多已经不能放下[1~r]了，最多只能放下[1~r-1]了。

```
int missNum(std::vector<int> &nums){
    int l = 0;
    int r = nums.size();
    while(l <= r ) {
        if(arr[l] == l+1) {
            l++;
        }else if(arr[l] <= l || arr[l] > r || arr[arr[l] - 1] == arr[l]){
            arr[l] = arr[--r];
        }else{
            std::swap(nums[l], nums[l]-1);
        }
    }

    return l+1;
}
```

8. 求需要排序的最小数组长度

分析：

- 题目要求升序
- 那么[maxIndex ... minIndex]之间的元素一定是失序的。

```
int getMinLength(std::vector<int> &nums) {
    int len = nums.size();
    int minIndex = -1;
    int mine = nums[len-1]; // 从后向前查找
    for(int i=len-2;i>=0;i--) {
        if(nums[i] > mine){
            minIndex = i; // 已经失序，存在min前的元素大于min，记录比min值大的元素，代表了失序开始的位置
        }else{
            mine = std::min(mine, nums[i]); // 未失序则将更新最小值
        }
    }

    if(minIndex == -1) {
        return 0;
    }

    int maxe = nums[0];
    int maxIndex = -1;

    for(int i=1;i<len;i++){
        if(arr[i] < maxe){
            maxIndex = i; // 最大元素后有比其小的元素，标记其位置，代表了其后元素都有序
        }else{
            maxe = std::max(maxe, nums[i]);
        }
    }
}
```

```
    return maxIndex - minIndex + 1;
}
```

9. 查找出现次数大于一般的元素

```
int getNum(std::vector<int> &nums) {
    int times = 0;
    int num = 0;
    int len = nums.size();

    for(int i=0;i<len;i++){
        if(0 == times) {
            times = 1;
            num = nums[i];
        } else if (nums[i] == num) {
            times++;
        } else{
            times--;
        }
    }
    return num;
}
```

10. 最大子数组和 给定一个数组，求最大的子数组和。

```
int getSum(std::vector<int> &nums) {
    int curr = 0;
    int maxsum = 0;
    int len = nums.size();
    int i = 0;

    while(i < len) {
        curr += nums[i];
        maxsum = std::max(maxsum, curr);
        if(curr < 0 ) {
            curr = 0;
        }
    }

    return maxsum;
}
```

11. 子数组的最大累加和问题 如果一个矩阵有k行，且限定必须有k行元素的情况下，我们只需要将矩阵的每一列的k个元素累加生成一个数组，然后求出这个数组的最大累加和,这个最大累加和就是 必须含有 k 行元素的子矩阵中的最大累加和。

```
int maxSum(std::vector<std::vector<int>> &nums){
    int row = nums.size();
    int col = nums[0].size();
    int maxv = INT_MIN;
    int cur = 0;
    std::vector<int> currsum(row) ; // 累加和数组
    for(int i=0;i < row;i++) {
        currsum.clear();
        for(int j=i;j < row;j++){
            cur = 0;
            for(int k=0;k < col;k++){
                s[k] += nums[j][k];
                curr += s[k];
                maxv = std::max(maxv, curr);
                if(curr < 0) {
                    curr = 0;
                }
            }
        }
    }

    return maxv == INT_MIN?0:maxv;
}
```

12. 局部最小 定义局部最小概念：

arr