221. 最大正方形

221. 最大正方形

难度 中等 296 收藏 分享 切换为英文 关注 反馈

在一个由 0 和 1 组成的二维矩阵内，找到只包含 1 的最大正方形，并返回其面积。

示例:

输入：

```
1 0 1 0 0
1 0 1 1 1
1 1 1 1 1
1 0 0 1 0
```

输出： 4

```
class Solution {
public:
    int maximalSquare(vector<vector<char>>& matrix) {
        if(matrix.size()==0)
            return 0;
        if(matrix[0].size()==0)
            return 0;
        int m = matrix.size();
        int n = matrix[0].size();
        vector<vector<int> > dp(m, vector<int>(n, 0));
        int res = 0;
        for(int i = 0;i<m;i++){
            for(int j  = 0;j<n;j++){
                if(matrix[i][j]=='0')
                    dp[i][j] = 0;
                else{
                    dp[i][j] = 1;
                    if(i>=1&&j>=1)
                        dp[i][j]+=min(dp[i-1][j],min(dp[i-1][j-1], dp[i][j-1]));
                    if(dp[i][j]>res)
                        res=dp[i][j];
                }
            }
        }
        return res*res;
    }
};
```

222. 完全二叉树的节点个数

222. 完全二叉树的节点个数

难度 中等 152 收藏 分享 切换为英文 关注 反馈

给出一个**完全二叉树**，求出该树的节点个数。

说明：

完全二叉树的定义如下：在完全二叉树中，除了最底层节点可能没填满外，其余每层节点数都达到最大值，并且最下面一层的节点都集中在该层最左边的若干位置。若最底层为第 h 层，则该层包含 1~ 2^h 个节点。

示例：

```
输入：
    1
   / \
  2   3
 / \  /
4  5 6

输出：6
```

```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    int countNodes(TreeNode* root) {
        if(!root)
            return 0;
        int l = 0;
        int r = 0;
        TreeNode* leftp = root;
        TreeNode* rightp = root;
        while(leftp){
            l++;
            leftp = leftp->left;
        }
        while(rightp){
            r++;
            rightp = rightp->right;
        }
        if(l==r)
            return (1<<l)-1;
        return countNodes(root->left) + countNodes(root->right) + 1;
    }
};
```

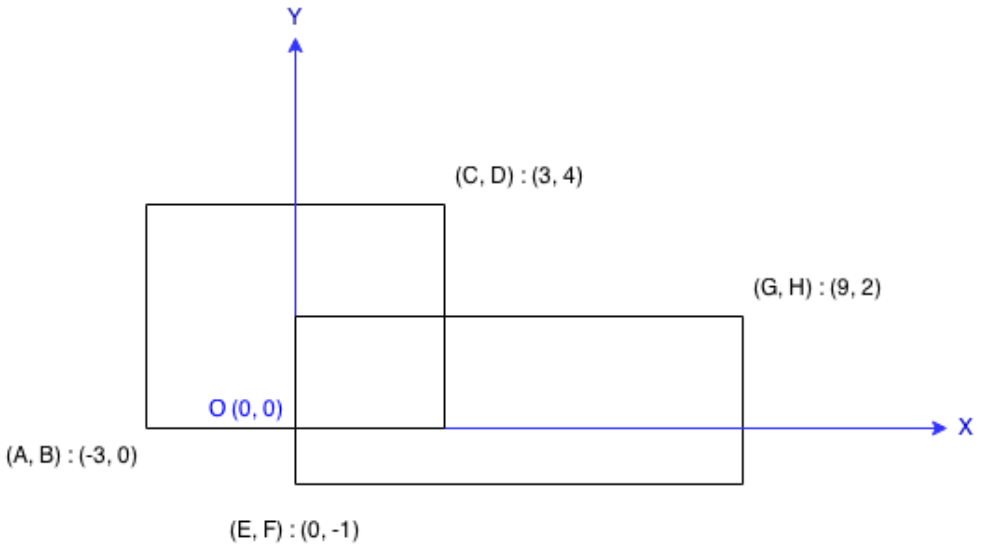
223. 矩形面积

223. 矩形面积

难度 中等 63 收藏 分享 切换为英文 关注 反馈

在**二维**平面上计算出两个**由直线构成的**矩形重叠后形成的总面积。

每个矩形由其左下顶点和右上顶点坐标表示，如图所示。



示例:

```
输入：-3, 0, 3, 4, 0, -1, 9, 2
输出：45
```

说明: 假设矩形面积不会超出 **int** 的范围。

```
class Solution {
public:
    int computeArea(int A, int B, int C, int D, int E, int F, int G, int H) {
        long long X = min(C, G) + 0ll - max(A, E);
        long long Y = min(D, H) + 0ll - max(B, F);
        return (C - A) * (D - B) - max(0ll, X) * max(0ll, Y) + (G - E) * (H -
F);
    }
};
```

224. 基本计算器

224. 基本计算器

难度 困难 181 收藏 分享 切换为英文 关注 反馈

实现一个基本的计算器来计算一个简单的字符串表达式的值。

字符串表达式可以包含左括号 `(` ，右括号 `)`，加号 `+` ，减号 `-`，**非负**整数和空格 ` ` 。

示例 1:

```
输入："1 + 1"
输出：2
```

示例 2:

```
输入：" 2-1 + 2 "
输出：3
```

示例 3:

```
输入："(1+(4+5+2)-3)+(6+8)"
输出：23
```

说明:

- 你可以假设所给定的表达式都是有效的。
- 请**不要**使用内置的库函数 `eval`。

```
class Solution {
public:

    void calc(stack<char> &op, stack<int> &num) {
        int y = num.top();
        num.pop();
        int x = num.top();
        num.pop();
        if (op.top() == '+') num.push(x + y);
        else num.push(x - y);
        op.pop();
    }

    int calculate(string s) {
        stack<char> op;
        stack<int> num;
        for (int i = 0; i < s.size(); i ++ ) {
            char c = s[i];
            if (c == ' ') continue;
            if (c == '+' || c == '-' || c == '(') op.push(c);
            else if (c == ')') {
                op.pop();
                if (op.size() && op.top() != '(') {
                    calc(op, num);
                }
            }
            else {
                int j = i;
                while (j < s.size() && isdigit(s[j])) j ++ ;
```

```
                num.push(atoi(s.substr(i, j - i).c_str()));
                i = j - 1;
                if (op.size() && op.top() != '(') {
                    calc(op, num);
                }
            }
        }
        return num.top();
    }
};
```

225. 用队列实现栈

225. 用队列实现栈

难度 简单 166 收藏 分享 切换为英文 关注 反馈

使用队列实现栈的下列操作：

- push(x) -- 元素 x 入栈
- pop() -- 移除栈顶元素
- top() -- 获取栈顶元素
- empty() -- 返回栈是否为空

注意:

- 你只能使用队列的基本操作-- 也就是 `push to back`, `peek/pop from front`, `size`, 和 `is empty` 这些操作是合法的。
- 你所使用的语言也许不支持队列。 你可以使用 list 或者 deque（双端队列）来模拟一个队列 , 只要是标准的队列操作即可。
- 你可以假设所有操作都是有效的（例如, 对一个空的栈不会调用 pop 或者 top 操作）。

```
class MyStack {
public:
    /** Initialize your data structure here. */
    queue<int> q;
    MyStack() {

    }

    /** Push element x onto stack. */
    void push(int x) {
        q.push(x);
    }

    /** Removes the element on top of the stack and returns that element. */
    int pop() {
        int sz = q.size();
        while (sz -- > 1) q.push(q.front()), q.pop();
        int x = q.front();
        q.pop();
        return x;
    }

    /** Get the top element. */
    int top() {
        int sz = q.size();
```

```
        while (sz -- > 1) q.push(q.front()), q.pop();
        int x = q.front();
        q.pop(), q.push(x);
        return x;
    }

    /** Returns whether the stack is empty. */
    bool empty() {
        return q.empty();
    }
};

/**
 * Your MyStack object will be instantiated and called as such:
 * MyStack obj = new MyStack();
 * obj.push(x);
 * int param_2 = obj.pop();
 * int param_3 = obj.top();
 * bool param_4 = obj.empty();
 */
```

226. 翻转二叉树

226. 翻转二叉树

难度 简单 417 收藏 分享 切换为英文 关注 反馈

翻转一棵二叉树。

示例:

输入:

```
     4
   /   \
  2     7
 / \   / \
1   3 6   9
```

输出:

```
     4
   /   \
  7     2
 / \   / \
9   6 3   1
```

备注:
这个问题是受到 Max Howell 的 原问题 启发的 :

> 谷歌: 我们90%的工程师使用您编写的软件(Homebrew), 但是您却无法在面试时在白板上写出翻转二叉树这道题, 这太糟糕了。

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
```

```
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    TreeNode* invertTree(TreeNode* root) {
        if (!root) return 0;
        swap(root->left, root->right);
        invertTree(root->left), invertTree(root->right);
        return root;
    }
};
```

227. 基本计算器 II

227. 基本计算器 II

难度 中等 125 收藏 分享 切换为英文 关注 反馈

实现一个基本的计算器来计算一个简单的字符串表达式的值。

字符串表达式仅包含非负整数，`+`， `-` ，`*`，`/` 四种运算符和空格 `  `。 整数除法仅保留整数部分。

示例 1:

```
输入: "3+2*2"
输出: 7
```

示例 2:

```
输入: " 3/2 "
输出: 1
```

示例 3:

```
输入: " 3+5 / 2 "
输出: 5
```

说明:

- 你可以假设所给定的表达式都是有效的。
- 请**不要**使用内置的库函数 `eval`。

```
class Solution {
```

```
public:
    int calculate(string s) {
        if(s.size()==0)
            return 0;
        int res = 0;
        stack<int>stk;
        int num = 0;
        char sign = '+';
        for(int i = 0;i<s.size();i++){
            if(s[i] >= '0' && s[i] <= '9'){
                num *= 10;
                num = num - '0' + s[i];
            }
            if((s[i]=='+'||s[i]=='-'||s[i]=='*'||s[i]=='/')||i==s.size()-1){
                if(sign=='+')
                    stk.push(num);
                if(sign=='-')
                    stk.push(-num);
                if(sign=='*'){
                    int topnum = stk.top();
                    stk.pop();
                    stk.push(num*topnum);
                }
                if(sign=='/'){
                    int topnum = stk.top();
                    stk.pop();
                    stk.push(topnum/num);
                }
                sign=s[i];
                num = 0;
            }
        }
        while(!stk.empty()){
            int topnum = stk.top();
            res += topnum;
            stk.pop();
        }
        return res;
    }
};
```

228. 汇总区间

228. 汇总区间

难度 中等 46 收藏 分享 切换为英文 关注 反馈

给定一个无重复元素的有序整数数组，返回数组区间范围的汇总。

示例 1:

```
输入：[0,1,2,4,5,7]
输出：["0->2","4->5","7"]
解释：0,1,2 可组成一个连续的区间；4,5 可组成一个连续的区间。
```

示例 2:

```
输入：[0,2,3,4,6,8,9]
输出：["0","2->4","6","8->9"]
解释：2,3,4 可组成一个连续的区间；8,9 可组成一个连续的区间。
```

```
class Solution {
public:
    vector<string> summaryRanges(vector<int>& nums) {
        vector<string> res;
        int st, ed;
        for (int i = 0; i < nums.size(); i ++ )
        {
            int x = nums[i];
            if (!i) st = ed = x;
            else if (x == ed + 1) ed ++ ;
            else
            {
                res.push_back(to_string(st) + (st == ed ? "" : "->" +
to_string(ed)));
                st = ed = x;
            }
        }
        if (nums.size()) res.push_back(to_string(st) + (st == ed ? "" : "->" +
to_string(ed)));
        return res;
    }
};
```

229. 求众数 II

229. 求众数 II

难度 中等 178 收藏 分享 切换为英文 关注 反馈

给定一个大小为 *n* 的数组，找出其中所有出现超过 `⌊ n/3 ⌋` 次的元素。

说明: 要求算法的时间复杂度为 O(n)，空间复杂度为 O(1)。

示例 1:

```
输入: [3,2,3]
输出: [3]
```

示例 2:

```
输入: [1,1,1,3,3,2,2,2]
输出: [1,2]
```

```cpp
class Solution {
public:
    vector<int> majorityElement(vector<int>& nums) {
        int numsSize = nums.size();
        vector<int>ret;
        if(numsSize == 0) return ret;
        int count1 = 0;
        int count2 = 0;
        int candidate1 = 0;
        int candidate2 = 0;
        for(auto i : nums)
        {//第一轮扫描确定candidate
            if(candidate1 == i) count1++;
            else if (candidate2 == i) count2++;
            else if (count1 == 0)
            {
                candidate1 = i;
                count1 = 1;
            }
            else if (count2 == 0)
            {
                candidate2 = i;
                count2 = 1;
            }
            else
            {
                count1--;
                count2--;
            }
        }
        count1 = 0;
        count2 = 0;
        for(auto i : nums)
        {//第二轮扫描确定candidate是否符合要求
            if(candidate1 == i) count1++;
            else if(candidate2 == i) count2++;
        }
        if(count1 > numsSize/3) ret.push_back(candidate1);
```

```
        if(count2 > numsSize/3) ret.push_back(candidate2);
        return ret;
    }
};
```

230. 二叉搜索树中第K小的元素

230. 二叉搜索树中第K小的元素

难度 中等 189 收藏 分享 切换为英文 关注 反馈

给定一个二叉搜索树，编写一个函数 `kthSmallest` 来查找其中第 **k** 个最小的元素。

说明:
你可以假设 k 总是有效的，1 ≤ k ≤ 二叉搜索树元素个数。

示例 1:

```
输入: root = [3,1,4,null,2], k = 1
   3
  / \
 1   4
  \
   2
输出: 1
```

示例 2:

```
输入: root = [5,3,6,2,4,null,null,1], k = 3
       5
      / \
     3   6
    / \
   2   4
  /
 1
输出: 3
```

进阶:
如果二叉搜索树经常被修改（插入/删除操作）并且你需要频繁地查找第 k 小的值，你将如何优化 `kthSmallest` 函数?

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    void dfs(TreeNode* cur, vector<TreeNode*>& nodeList) {
        if (cur -> left != NULL)
```

```
            dfs(cur -> left, nodeList);
        nodeList.push_back(cur);
        if (cur -> right != NULL)
            dfs(cur -> right, nodeList);
    }
    int kthSmallest(TreeNode* root, int k) {
        vector<TreeNode*> nodeList;
        dfs(root, nodeList);
        return nodeList[k - 1] -> val;
    }
};
```

231. 2的幂

231. 2的幂

难度 简单 185 收藏 分享 切换为英文 关注 反馈

给定一个整数，编写一个函数来判断它是否是 2 的幂次方。

示例 1:

```
输入: 1
输出: true
解释: 2^0 = 1
```

示例 2:

```
输入: 16
输出: true
解释: 2^4 = 16
```

示例 3:

```
输入: 218
输出: false
```

```
class Solution {
public:
    bool isPowerOfTwo(int n) {
        return (n > 0) ? ((n & -n) == n) : false;
    }
};
```

232. 用栈实现队列

232. 用栈实现队列

难度 简单 162 收藏 分享 切换为英文 关注 反馈

使用栈实现队列的下列操作：

- push(x) -- 将一个元素放入队列的尾部。
- pop() -- 从队列首部移除元素。
- peek() -- 返回队列首部的元素。
- empty() -- 返回队列是否为空。

示例:

```
MyQueue queue = new MyQueue();

queue.push(1);
queue.push(2);
queue.peek();  // 返回 1
queue.pop();   // 返回 1
queue.empty(); // 返回 false
```

说明:

- 你只能使用标准的栈操作 -- 也就是只有 `push to top`, `peek/pop from top`, `size`, 和 `is empty` 操作是合法的。
- 你所使用的语言也许不支持栈。你可以使用 list 或者 deque（双端队列）来模拟一个栈，只要是标准的栈操作即可。
- 假设所有操作都是有效的 （例如，一个空的队列不会调用 pop 或者 peek 操作）。

```
class MyQueue {
public:
    /** Initialize your data structure here. */
    stack<int>sta, cache;
    MyQueue() {

    }

    /** Push element x to the back of queue. */
    void push(int x) {
        sta.push(x);
    }

    /** Removes the element from in front of queue and returns that element. */
    int pop() {
        while (!sta.empty()) cache.push(sta.top()), sta.pop();
        int x = cache.top();
        cache.pop();
        while (!cache.empty()) sta.push(cache.top()), cache.pop();
        return x;
    }

    /** Get the front element. */
    int peek() {
        while (!sta.empty()) cache.push(sta.top()), sta.pop();
        int x = cache.top();
        while (!cache.empty()) sta.push(cache.top()), cache.pop();
        return x;
    }
```

```
    /** Returns whether the queue is empty. */
    bool empty() {
        return sta.empty();
    }
};

/**
 * Your MyQueue object will be instantiated and called as such:
 * MyQueue obj = new MyQueue();
 * obj.push(x);
 * int param_2 = obj.pop();
 * int param_3 = obj.peek();
 * bool param_4 = obj.empty();
 */
```

233. 数字 1 的个数

233. 数字 1 的个数

难度 困难 122 收藏 分享 切换为英文 关注 反馈

给定一个整数 n，计算所有小于等于 n 的非负整数中数字 1 出现的个数。

示例:

```
输入：13
输出：6
解释：数字 1 出现在以下数字中：1, 10, 11, 12, 13 。
```

```
class Solution {
public:
    int countDigitOne(int n) {
        if (n <= 0)
            return 0;

        vector<int> f(10, 0), pow(10, 0);
        f[0] = 0;
        pow[0] = 1;
        for (int i = 1; i <= 9; i++) {
            f[i] = f[i - 1] * 10 + pow[i - 1];
            pow[i] = pow[i - 1] * 10;
        }
        string num = to_string(n);
        reverse(num.begin(), num.end());
        int ans = 0, ones = 0;
        for (int i = num.length() - 1; i >= 0; i--) {
            ans += ones * ((num[i] - '0') * pow[i]);
            if (num[i] == '1') {
                ones++;
                ans += f[i];
            }
            else if (num[i] != '0')
                ans += pow[i] + f[i] * (num[i] - '0');
        }
        return ans + ones;
```

```
    }
};
```

234. 回文链表

234. 回文链表

难度 简单 475 收藏 分享 切换为英文 关注 反馈

请判断一个链表是否为回文链表。

示例 1:

```
输入: 1->2
输出: false
```

示例 2:

```
输入: 1->2->2->1
输出: true
```

进阶:
你能否用 O(n) 时间复杂度和 O(1) 空间复杂度解决此题?

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    bool isPalindrome(ListNode* head) {
        int n = 0;
        for (ListNode *p = head; p; p = p->next) n ++ ;
        if (n <= 1) return true;
        ListNode *a = head;
        for (int i = 0; i < (n + 1) / 2 - 1; i ++ ) a = a->next;
        ListNode *b = a->next;
        while (b)
        {
            ListNode *c = b->next;
            b->next = a;
            a = b;
            b = c;
        }
        b = head;
        ListNode *tail = a;
        bool res = true;
        for (int i = 0; i < n / 2; i ++ )
        {
            if (a->val != b->val)
            {
                res = false;
```

```
                break;
            }
            a = a->next;
            b = b->next;
        }
        a = tail, b = a->next;
        for (int i = 0; i < n / 2; i ++ )
        {
            ListNode *c = b->next;
            b->next = a;
            a = b;
            b = c;
        }
        tail->next = 0;
        return res;
    }
};
```

235. 二叉搜索树的最近公共祖先

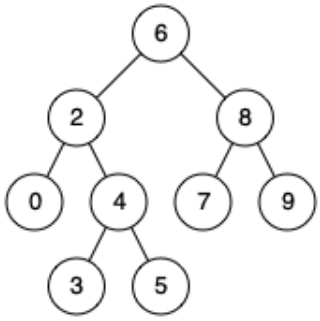
235. 二叉搜索树的最近公共祖先

难度 简单 269 收藏 分享 切换为英文 关注 反馈

给定一个二叉搜索树, 找到该树中两个指定节点的最近公共祖先。

百度百科中最近公共祖先的定义为：“对于有根树 T 的两个结点 p、q，最近公共祖先表示为一个结点 x，满足 x 是 p、q 的祖先且 x 的深度尽可能大（**一个节点也可以是它自己的祖先**）。”

例如，给定如下二叉搜索树: root = [6,2,8,0,4,7,9,null,null,3,5]



示例 1:

```
输入: root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 8
输出: 6
解释: 节点 2 和节点 8 的最近公共祖先是 6。
```

示例 2:

```
输入: root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 4
输出: 2
解释: 节点 2 和节点 4 的最近公共祖先是 2, 因为根据定义最近公共祖先节点可以为节点本身。
```

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
        TreeNode *cur = root;
        while (1) {
            if (p -> val < cur -> val && q -> val < cur -> val)
                cur = cur -> left;
            else if (p -> val > cur -> val && q -> val > cur -> val)
                cur = cur -> right;
            else
                break;
        }
        return cur;
    }
};
```

236. 二叉树的最近公共祖先

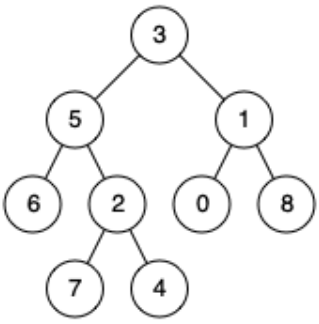
236. 二叉树的最近公共祖先

难度 中等 472 收藏 分享 切换为英文 关注 反馈

给定一个二叉树, 找到该树中两个指定节点的最近公共祖先。

百度百科中最近公共祖先的定义为：“对于有根树 T 的两个结点 p、q，最近公共祖先表示为一个结点 x，满足 x 是 p、q 的祖先且 x 的深度尽可能大（**一个节点也可以是它自己的祖先**）。”

例如，给定如下二叉树: root = [3,5,1,6,2,0,8,null,null,7,4]



示例 1:

```
输入: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 1
输出: 3
解释: 节点 5 和节点 1 的最近公共祖先是节点 3。
```

示例 2:

```
输入: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 4
输出: 5
解释: 节点 5 和节点 4 的最近公共祖先是节点 5。因为根据定义最近公共祖先节点可以为节点本身。
```

```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
        if(!root || root == p || root == q) return root;

        auto left = lowestCommonAncestor(root->left,p,q);
        auto right = lowestCommonAncestor(root->right,p,q);

        if(!left) return right;
        if(!right) return left;
        return root;
    }
```

```
};
```

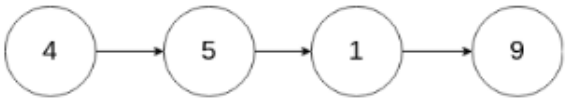
237. 删除链表中的节点

237. 删除链表中的节点

难度 简单 662 收藏 分享 切换为英文 关注 反馈

请编写一个函数，使其可以删除某个链表中给定的（非末尾）节点，你将只被给定要求被删除的节点。

现有一个链表 -- head = [4,5,1,9]，它可以表示为:



示例 1:

```
输入: head = [4,5,1,9], node = 5
输出: [4,1,9]
解释: 给定你链表中值为 5 的第二个节点，那么在调用了你的函数之后，该链表应变为 4 -
> 1 -> 9.
```

示例 2:

```
输入: head = [4,5,1,9], node = 1
输出: [4,5,9]
解释: 给定你链表中值为 1 的第三个节点，那么在调用了你的函数之后，该链表应变为 4 -
> 5 -> 9.
```

说明:

- 链表至少包含两个节点。
- 链表中所有节点的值都是唯一的。
- 给定的节点为非末尾节点并且一定是链表中的一个有效节点。
- 不要从你的函数中返回任何结果。

```cpp
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    void deleteNode(ListNode* node) {
        node->val = node->next->val;
        node->next = node->next->next;
    }
};
```

238. 除自身以外数组的乘积

238. 除自身以外数组的乘积

难度 中等 374 收藏 分享 切换为英文 关注 反馈

给你一个长度为 *n* 的整数数组 `nums`，其中 *n* > 1，返回输出数组 `output` ，其中 `output[i]` 等于 `nums` 中除 `nums[i]` 之外其余各元素的乘积。

示例:

```
输入：[1,2,3,4]
输出：[24,12,8,6]
```

提示： 题目数据保证数组之中任意元素的全部前缀元素和后缀（甚至是整个数组）的乘积都在 32 位整数范围内。

说明: 请**不要使用除法，**且在 O(*n*) 时间复杂度内完成此题。

进阶:
你可以在常数空间复杂度内完成这个题目吗？（出于对空间复杂度分析的目的，输出数组**不被视为**额外空间。）

```cpp
class Solution {
public:
    vector<int> productExceptSelf(vector<int>& nums) {
        int n = nums.size();
        vector<int> output(n, 1);
        for (int i = 1; i < n; i++)
            output[i] = output[i - 1] * nums[i - 1];

        int end = 1;
        for (int i = n - 1; i >= 0; i--) {
            output[i] *= end;
            end *= nums[i];
        }
        return output;
    }
};
```

239. 滑动窗口最大值

239. 滑动窗口最大值

难度 困难 336 收藏 分享 切换为英文 关注 反馈

给定一个数组 *nums*，有一个大小为 *k* 的滑动窗口从数组的最左侧移动到数组的最右侧。你只可以看到在滑动窗口内的 *k* 个数字。滑动窗口每次只向右移动一位。

返回滑动窗口中的最大值。

进阶：

你能在线性时间复杂度内解决此题吗？

示例：

```
输入：nums = [1,3,-1,-3,5,3,6,7], 和 k = 3
输出：[3,3,5,5,6,7]
解释：

  滑动窗口的位置                最大值
---------------               -----
[1  3  -1] -3  5  3  6  7       3
 1 [3  -1  -3] 5  3  6  7       3
 1  3 [-1  -3  5] 3  6  7       5
 1  3  -1 [-3  5  3] 6  7       5
 1  3  -1  -3 [5  3  6] 7       6
 1  3  -1  -3  5 [3  6  7]      7
```

提示：

- `1 <= nums.length <= 10^5`
- `-10^4 <= nums[i] <= 10^4`
- `1 <= k <= nums.length`

```cpp
class Solution {
public:
    vector<int> maxSlidingWindow(vector<int>& nums, int k) {
        vector<int> res;
        deque<int> q;
        for(int i = 0;i < nums.size();i ++)
        {
            if(q.size() && i - k + 1 > q.front()) q.pop_front();
            while(q.size() && nums[q.back()] <= nums[i]) q.pop_back();
            q.push_back(i);
            if(i >= k - 1) res.push_back(nums[q.front()]);
        }
        return res;
    }
};
```

240. 搜索二维矩阵 II

240. 搜索二维矩阵 II

难度 中等 286 收藏 分享 切换为英文 关注 反馈

编写一个高效的算法来搜索 *m* x *n* 矩阵 matrix 中的一个目标值 target。该矩阵具有以下特性：

- 每行的元素从左到右升序排列。
- 每列的元素从上到下升序排列。

示例:

现有矩阵 matrix 如下：

```
[
  [1,   4,  7, 11, 15],
  [2,   5,  8, 12, 19],
  [3,   6,  9, 16, 22],
  [10, 13, 14, 17, 24],
  [18, 21, 23, 26, 30]
]
```

给定 target = `5`，返回 `true`。

给定 target = `20`，返回 `false`。

```cpp
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        int m = matrix.size();
        if (m == 0)
            return false;
        int n = matrix[0].size();

        int up = 0, right = n - 1;
        while (up < m && right >= 0) {
            if (matrix[up][right] == target)
                return true;
            else if (matrix[up][right] < target)
                up++;
            else
                right--;
        }

        return false;
    }
};
```