283. 移动零

283. 移动零

难度 简单 571 收藏 分享 切换为英文 关注 反馈

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。

```
class Solution {
public:
    void moveZeroes(vector<int>& nums) {
        int i = 0;
        for (int j = 0; j < nums.size(); j++)
            if (nums[j] != 0)
                nums[i++] = nums[j];
        for (; i < nums.size(); i++)
            nums[i] = 0;
    }
};
```

284. 顶端迭代器

284. 顶端迭代器

难度 中等 45 收藏 分享 切换为英文 关注 反馈

给定一个迭代器类的接口，接口包含两个方法： `next()` 和 `hasNext()`。设计并实现一个支持 `peek()` 操作的顶端迭代器 -- 其本质就是把原本应由 `next()` 方法返回的元素 `peek()` 出来。

示例:

```
假设迭代器被初始化为列表 [1,2,3]。

调用 next() 返回 1，得到列表中的第一个元素。
现在调用 peek() 返回 2，下一个元素。在此之后调用 next() 仍然返回 2。
最后一次调用 next() 返回 3，末尾元素。在此之后调用 hasNext() 应该返回 false。
```

进阶: 你将如何拓展你的设计？使之变得通用化，从而适应所有的类型，而不只是整数型？

```
// Below is the interface for Iterator, which is already defined for you.
// **DO NOT** modify the interface for Iterator.
class Iterator {
```

```
    struct Data;
    Data* data;
public:
    Iterator(const vector<int>& nums);
    Iterator(const Iterator& iter);
    virtual ~Iterator();
    // Returns the next element in the iteration.
    int next();
    // Returns true if the iteration has more elements.
    bool hasNext() const;
};


class PeekingIterator : public Iterator {
public:
    int _next;
    bool _has_next;

    PeekingIterator(const vector<int>& nums) : Iterator(nums) {
        // Initialize any member here.
        // **DO NOT** save a copy of nums and manipulate it directly.
        // You should only use the Iterator interface methods.
        _has_next = Iterator::hasNext();
        if (_has_next)
            _next = Iterator::next();
    }

    // Returns the next element in the iteration without advancing the iterator.
    int peek() {
        return _next;
    }

    // hasNext() and next() should behave the same as in the Iterator interface.
    // Override them if needed.
    int next() {
        int res = _next;
        _has_next = Iterator::hasNext();
        if (_has_next)
            _next = Iterator::next();
        return res;
    }

    bool hasNext() const {
        return _has_next;
    }
};
```

287. 寻找重复数

287. 寻找重复数

难度 中等 538 收藏 分享 切换为英文 关注 反馈

给定一个包含 *n* + 1 个整数的数组 *nums*，其数字都在 1 到 *n* 之间（包括 1 和 *n*），可知至少存在一个重复的整数。假设只有一个重复的整数，找出这个重复的数。

示例 1:

```
输入：[1,3,4,2,2]
输出：2
```

示例 2:

```
输入：[3,1,3,4,2]
输出：3
```

说明:

1. **不能**更改原数组（假设数组是只读的）。
2. 只能使用额外的 *O*(1) 的空间。
3. 时间复杂度小于 $O(n^2)$ 。
4. 数组中只有一个重复的数字，但它可能不止重复出现一次。

```
class Solution {
public:
    int findDuplicate(vector<int>& nums) {
        int l = 1,r = nums.size() - 1;
        while(l < r)
        {
            int mid = l + r >> 1;
            int cnt = 0;
            for(auto x : nums)
                if(x >= l && x <= mid)
                    cnt ++;

            if(cnt > mid - l + 1) r = mid;
            else l = mid + 1;
        }
        return r;
    }
};
```

289. 生命游戏

289. 生命游戏

难度 中等 199 收藏 分享 切换为英文 关注 反馈

根据 百度百科 ，生命游戏，简称为生命，是英国数学家约翰·何顿·康威在 1970 年发明的细胞自动机。

给定一个包含 m × n 个格子的面板，每一个格子都可以看成是一个细胞。每个细胞都具有一个初始状态：1 即为活细胞（live），或 0 即为死细胞（dead）。每个细胞与其八个相邻位置（水平，垂直，对角线）的细胞都遵循以下四条生存定律：

1. 如果活细胞周围八个位置的活细胞数少于两个，则该位置活细胞死亡；
2. 如果活细胞周围八个位置有两个或三个活细胞，则该位置活细胞仍然存活；
3. 如果活细胞周围八个位置有超过三个活细胞，则该位置活细胞死亡；
4. 如果死细胞周围正好有三个活细胞，则该位置死细胞复活；

根据当前状态，写一个函数来计算面板上所有细胞的下一个（一次更新后的）状态。下一个状态是通过将上述规则同时应用于当前状态下的每个细胞所形成的，其中细胞的出生和死亡是同时发生的。

示例：

输入：

```
[
  [0,1,0],
  [0,0,1],
  [1,1,1],
  [0,0,0]
]
```

输出：

```
[
  [0,0,0],
  [1,0,1],
  [0,1,1],
  [0,1,0]
]
```

```
class Solution {
public:
    void gameOfLife(vector<vector<int>>& board) {
        int m = board.size();
        if(m==0)
            return;
        int n = board[0].size();
        for(int i = 0;i < m;i++){
            for(int j = 0;j < n;j++){
                int neighbour = 0;//记录周围八个网格的活细胞数
                for(int di = -1;di <= 1; di++){
                    for(int dj = -1;dj <= 1;dj++){
                        if(i + di >=0 && i+di < m && j+dj >= 0 && j+dj < n &&!
(di == 0 && dj == 0)){
                            neighbour += board[i+di][j+dj] & 1;//取每个网格的最低位
（当前状态）
                        }
                    }
                }
                    if(board[i][j] == 1){
                        if(neighbour < 2 || neighbour > 3)
```

```
                            board[i][j] = 1;//更新后状态为0，所以记为01（二进制）
                        else
                            board[i][j] = 3;//更新后状态为1，所以记为11（二进制）
                    }
                    else{
                        if(neighbour == 3)
                            board[i][j]=2;//更新后状态为1，所以记为10（二进制）
                        else
                            board[i][j] = 0;//更新后状态为0，所以记为00（二进制）
                    }
                }
            }

        for(int i = 0;i<m;i++){
            for(int j = 0;j<n;j++){
                board[i][j] =(board[i][j]&2)>>1;//取每个网格的第二位，得到更新后状态
            }
        }
    }
};
```

290. 单词规律

290. 单词规律

难度 简单 147 收藏 分享 切换为英文 关注 反馈

给定一种规律 `pattern` 和一个字符串 `str` ，判断 `str` 是否遵循相同的规律。

这里的 **遵循** 指完全匹配，例如， `pattern` 里的每个字母和字符串 `str` 中的每个非空单词之间存在着双向连接的对应规律。

示例1:

```
输入: pattern = "abba", str = "dog cat cat dog"
输出: true
```

示例 2:

```
输入:pattern = "abba", str = "dog cat cat fish"
输出: false
```

示例 3:

```
输入: pattern = "aaaa", str = "dog cat cat dog"
输出: false
```

示例 4:

```
输入: pattern = "abba", str = "dog dog dog dog"
输出: false
```

说明:
你可以假设 `pattern` 只包含小写字母， `str` 包含了由单个空格分隔的小写字母。

```cpp
class Solution {
public:
    bool wordPattern(string pattern, string str) {
        stringstream raw(str);
        vector<string> strs;
        string line;
        while (raw >> line) strs.push_back(line);
        if (pattern.size() != strs.size()) return false;
        unordered_map<char, string> PS;
        unordered_map<string, char> SP;
        for (int i = 0; i < pattern.size(); i ++ )
        {
            if (PS.count(pattern[i]) == 0) PS[pattern[i]] = strs[i];
            if (SP.count(strs[i]) == 0) SP[strs[i]] = pattern[i];
            if (PS[pattern[i]] != strs[i]) return false;
            if (SP[strs[i]] != pattern[i]) return false;
        }
        return true;
    }
};
```

292. Nim 游戏

292. Nim 游戏

难度 简单 296 收藏 分享 切换为英文 关注 反馈

你和你的朋友，两个人一起玩 Nim 游戏：桌子上有一堆石头，每次你们轮流拿掉 1 - 3 块石头。 拿掉最后一块石头的人就是获胜者。你作为先手。

你们是聪明人，每一步都是最优解。 编写一个函数，来判断你是否可以在给定石头数量的情况下赢得游戏。

示例:

```
输入：4
输出：false
解释：如果堆中有 4 块石头，那么你永远不会赢得比赛；
     因为无论你拿走 1 块、2 块 还是 3 块石头，最后一块石头总是会被你的朋友拿走。
```

```cpp
class Solution {
public:
    bool canWinNim(int n) {
        return n&3;
    }
};
```

295. 数据流的中位数

295. 数据流的中位数

难度 困难 169 收藏 分享 切换为英文 关注 反馈

中位数是有序列表中间的数。如果列表长度是偶数，中位数则是中间两个数的平均值。

例如，

[2,3,4] 的中位数是 3

[2,3] 的中位数是 (2 + 3) / 2 = 2.5

设计一个支持以下两种操作的数据结构：

- void addNum(int num) - 从数据流中添加一个整数到数据结构中。
- double findMedian() - 返回目前所有元素的中位数。

示例：

```
addNum(1)
addNum(2)
findMedian() -> 1.5
addNum(3)
findMedian() -> 2
```

进阶：

1. 如果数据流中所有整数都在 0 到 100 范围内，你将如何优化你的算法？
2. 如果数据流中 99% 的整数都在 0 到 100 范围内，你将如何优化你的算法？

```
class MedianFinder {
public:
    /** initialize your data structure here. */
    priority_queue<int> down;
    priority_queue<int,vector<int>,greater<int>> up;
    MedianFinder() {

    }

    void addNum(int num) {
        if(down.empty() || num >= down.top()) up.push(num);
        else
        {
            down.push(num);
            up.push(down.top());
            down.pop();
        }
        if(up.size() > down.size() + 1)
        {
            down.push(up.top());
            up.pop();
        }
    }

    double findMedian() {
        if(down.size() + up.size() & 1) return up.top();
        else return (down.top() + up.top()) / 2.0;
    }
};
```

```
/**
 * Your MedianFinder object will be instantiated and called as such:
 * MedianFinder* obj = new MedianFinder();
 * obj->addNum(num);
 * double param_2 = obj->findMedian();
 */
```

297. 二叉树的序列化与反序列化

297. 二叉树的序列化与反序列化

难度 困难 177 收藏 分享 切换为英文 关注 反馈

序列化是将一个数据结构或者对象转换为连续的比特位的操作，进而可以将转换后的数据存储在一个文件或者内存中，同时也可以通过网络传输到另一个计算机环境，采取相反方式重构得到原数据。

请设计一个算法来实现二叉树的序列化与反序列化。这里不限定你的序列 / 反序列化算法执行逻辑，你只需要保证一个二叉树可以被序列化为一个字符串并且将这个字符串反序列化为原始的树结构。

示例:

你可以将以下二叉树：

```
    1
   / \
  2   3
     / \
    4   5
```

序列化为 `"[1,2,3,null,null,4,5]"`

提示: 这与 LeetCode 目前使用的方式一致，详情请参阅 LeetCode 序列化二叉树的格式。你并非必须采取这种方式，你也可以采用其他的方法解决这个问题。

说明: 不要使用类的成员 / 全局 / 静态变量来存储状态，你的序列化和反序列化算法应该是无状态的。

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Codec {
public:

    // Encodes a tree to a single string.
    string serialize(TreeNode* root) {
        string res;
        dfs1(root,res);
        return res;
    }
    void dfs1(TreeNode *root,string &res)
    {
```

```
        if(!root)
        {
            res += "#,";
            return;
        }

        res += to_string(root->val) + ',';
        dfs1(root->left,res);
        dfs1(root->right,res);
    }
    // Decodes your encoded data to tree.
    TreeNode* deserialize(string data) {
        int u = 0;
        return dfs2(data,u);

    }
    TreeNode* dfs2(string &data,int &u)
    {
        if(data[u] == '#')
        {
            u += 2;
            return NULL;
        }

        int t = 0;
        bool is_minus = false;
        if(data[u] == '-')
        {
            is_minus = true;
            u ++;
        }
        while(data[u] != ',')
        {
            t = t * 10 + data[u] - '0';
            u ++;
        }
        u ++;
        if(is_minus) t = -t;

        auto root = new TreeNode(t);
        root->left = dfs2(data,u);
        root->right = dfs2(data,u);

        return root;

    }
};

// Your Codec object will be instantiated and called as such:
// Codec codec;
// codec.deserialize(codec.serialize(root));
```

299. 猜数字游戏

299. 猜数字游戏

难度 简单 67 收藏 分享 切换为英文 关注 反馈

你正在和你的朋友玩 猜数字（Bulls and Cows）游戏：你写下一个数字让你的朋友猜。每次他猜测后，你给他一个提示，告诉他有多少位数字和确切位置都猜对了（称为“Bulls”，公牛），有多少位数字猜对了但是位置不对（称为“Cows”，奶牛）。你的朋友将会根据提示继续猜，直到猜出秘密数字。

请写出一个根据秘密数字和朋友的猜测数返回提示的函数，用 `A` 表示公牛，用 `B` 表示奶牛。

请注意秘密数字和朋友的猜测数都可能含有重复数字。

示例 1:

```
输入: secret = "1807", guess = "7810"

输出: "1A3B"

解释: 1 公牛和 3 奶牛。公牛是 8，奶牛是 0, 1 和 7。
```

示例 2:

```
输入: secret = "1123", guess = "0111"

输出: "1A1B"

解释: 朋友猜测数中的第一个 1 是公牛，第二个或第三个 1 可被视为奶牛。
```

说明: 你可以假设秘密数字和朋友的猜测数都只包含数字，并且它们的长度永远相等。

```cpp
class Solution {
public:
    string getHint(string secret, string guess) {
        int n = secret.length(), bulls = 0, cows = 0;
        vector<int> vis(10, 0);
        for (int i = 0; i < n; i++) {
            vis[secret[i] - '0']++;
            if (secret[i] == guess[i])
                bulls++;
        }
        for (int i = 0; i < n; i++)
            if (vis[guess[i] - '0'] > 0) {
                cows++;
                vis[guess[i] - '0']--;
            }
        return to_string(bulls) + "A" + to_string(cows - bulls) + "B";
    }
};
```

300. 最长上升子序列

300. 最长上升子序列

难度 中等 668 收藏 分享 切换为英文 关注 反馈

给定一个无序的整数数组，找到其中最长上升子序列的长度。

示例:

```
输入：[10,9,2,5,3,7,101,18]
输出：4
解释：最长的上升子序列是 [2,3,7,101]，它的长度是 4。
```

说明:

- 可能会有多种最长上升子序列的组合，你只需要输出对应的长度即可。
- 你算法的时间复杂度应该为 $O(n^2)$ 。

进阶: 你能将算法的时间复杂度降低到 $O(n \log n)$ 吗?

```cpp
class Solution {
public:
    int lengthOfLIS(vector<int>& nums) {
        int n = nums.size();
        vector<int> f(n);

        for(int i = 0;i < n;i ++)
        {
            f[i] = 1;
            for(int j = 0;j < i;j ++)
                if(nums[j] < nums[i])
                    f[i] = max(f[i],f[j] + 1);
        }

        int res = 0;
        for(int i = 0;i < n;i ++) res = max(res,f[i]);
        return res;
    }
};
```