136. 只出现一次的数字（简单）

1. 题目描述

   给定一个非空整数数组，除了某个元素只出现一次以外，其余每个元素均出现两次。找出那个只出现了一次的元素。

   说明：你的算法应该具有线性时间复杂度。 你可以不使用额外空间来实现吗?

```
示例 1:
输入: [2,2,1]
输出: 1

示例 2:
输入: [4,1,2,1,2]
输出: 4
```

2. 简单实现

```
class Solution {
public:
    int singleNumber(vector<int>& nums) {
        int ans = nums[0];
        for(int i = 1; i < nums.size(); i++)
            ans ^= nums[i];
        return ans;
    }
};
```

19. 删除链表的倒数第N个节点（中等）

1. 题目描述

   给定一个链表，删除链表的倒数第 n 个节点，并且返回链表的头结点。

```
示例:
给定一个链表: 1->2->3->4->5, 和 n = 2.
当删除了倒数第二个节点后，链表变为 1->2->3->5.
```

   说明：给定的 n 保证是有效的。

   进阶： 你能尝试使用一趟扫描实现吗?

2. 简单实现

```
class Solution {
public:
    ListNode* removeNthFromEnd(ListNode* head, int n) {
```

```
        ListNode* h = new ListNode(0);//添加头节点，避免第一个节点被删除时的特殊处理
        h->next = head;
        ListNode* p = head;
        ListNode* q = h;
        while(n--)
            p = p->next;
        while(p){
            p = p->next;
            q = q->next;
        }
        q->next = q->next->next;
        return h->next;
    }
};
```

307. 区域和检索-数组可修改（中等）

1. 题目描述

   给定一个整数数组 nums，求出数组从索引 i 到 j (i ≤ j) 范围内元素的总和，包含 i, j 两点。

   update(i, val) 函数可以通过将下标为 i 的数值更新为 val，从而对数列进行修改。

   ```
   示例:
   Given nums = [1, 3, 5]
   sumRange(0, 2) -> 9
   update(1, 2)
   sumRange(0, 2) -> 8
   ```

   说明:

   - 数组仅可以在 update 函数下进行修改。
   - 你可以假设 update 函数与 sumRange 函数的调用次数是均匀分布的。

2. 简单实现

   树状数组，注意题目中update是更新值而不是增加值，所以还要保存数组每个元素的值。此外要牢记树状数组下标是以1开始的

```
class NumArray {
public:
    vector<int> nums, tree;
    int size;
    int lowbit(int x){
        return x & (-x);
    }
    NumArray(vector<int>& nums) {
        size = nums.size() + 1;//树状数组下标以1开始
        //初始化nums和trees
        this->nums = vector<int>(size, 0);
        tree = vector<int>(size, 0);
        for(int i = 0; i < size-1; i++)
            update(i, nums[i]);
    }
```

```
    void update(int i, int val) {
        i++;//下标对应
        val -= nums[i];//变化值
        nums[i] += val;//更新值
        while(i < size){
            tree[i] += val;
            i += lowbit(i);
        }
    }
    int getsum(int pos){
        int sum=0;
        pos++;//下标对应
        while(pos > 0){
            sum += tree[pos];
            pos -= lowbit(pos);
        }
        return sum;
    }
    int sumRange(int i, int j) {
        return getsum(j) - getsum(i-1);
    }

};
```

568. 最大休假天数（困难）

要会员

116. 填充每个节点的下一个右侧节点指针（中等）

1. 题目描述

   给定一个完美二叉树，其所有叶子节点都在同一层，每个父节点都有两个子节点。二叉树定义如下：

```
struct Node {
  int val;
  Node *left;
  Node *right;
  Node *next;
}
```

   填充它的每个 next 指针，让这个指针指向其下一个右侧节点。如果找不到下一个右侧节点，则将 next 指针设置为 NULL。

   初始状态下，所有 next 指针都被设置为 NULL。

   示例：

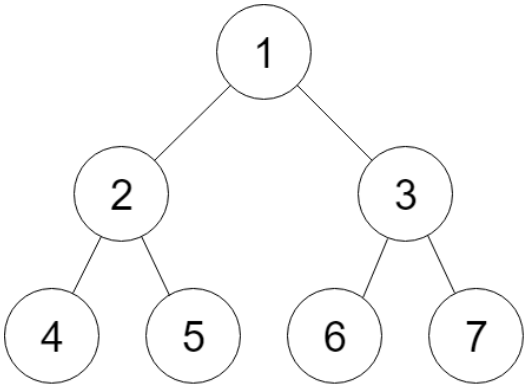


Figure A

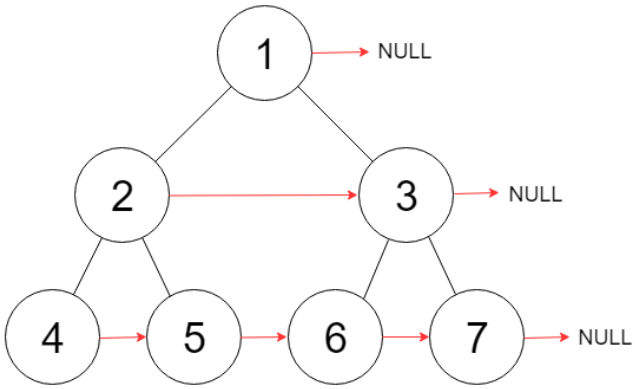


Figure B

```
输入：{"$id":"1","left":{"$id":"2","left":
{"$id":"3","left":null,"next":null,"right":null,"val":4},"next":null,"right":
{"$id":"4","left":null,"next":null,"right":null,"val":5},"val":2},"next":null,"righ
t":{"$id":"5","left":
{"$id":"6","left":null,"next":null,"right":null,"val":6},"next":null,"right":
{"$id":"7","left":null,"next":null,"right":null,"val":7},"val":3},"val":1}
输出：{"$id":"1","left":{"$id":"2","left":{"$id":"3","left":null,"next":
{"$id":"4","left":null,"next":{"$id":"5","left":null,"next":
{"$id":"6","left":null,"next":null,"right":null,"val":7},"right":null,"val":6},"rig
ht":null,"val":5},"right":null,"val":4},"next":{"$id":"7","left":
{"$ref":"5"},"next":null,"right":{"$ref":"6"},"val":3},"right":
{"$ref":"4"},"val":2},"next":null,"right":{"$ref":"7"},"val":1}
解释：给定二叉树如图 A 所示，你的函数应该填充它的每个 next 指针，以指向其下一个右侧节点，如图 B 所示。
```

提示：

- 你只能使用常量级额外空间。
- 使用递归解题也符合要求，本题中递归程序占用的栈空间不算做额外的空间复杂度

2. 简单实现

仿层序遍历

```
class Solution {
public:
    Node* connect(Node* root) {
        if (!root || !root->left)return root;
        root->left->next = root->right;
        Node* cur = root->left->left;
        Node* pre = root->left;
        while (cur!=NULL) {
            while (pre!=NULL) {
                pre->left->next = pre->right;
                if (pre->next != NULL)
                    pre->right->next = pre->next->left;
                pre = pre->next;
            }
            pre = cur;
            cur = cur->left;
        }
```

```
            return root;
        }
    };
```

268. 缺失数字（简单）

1. 题目描述

   给定一个包含 0, 1, 2, ..., n 中 n 个数的序列，找出 0 .. n 中没有出现在序列中的那个数。

   ```
   示例 1:
   输入: [3,0,1]
   输出: 2

   示例 2:
   输入: [9,6,4,2,3,5,7,0,1]
   输出: 8
   ```

   说明: 你的算法应具有线性时间复杂度。你能否仅使用额外常数空间来实现?

2. 简单实现

   鸽巢，修改了原数组

   ```
   class Solution {
   public:
       int missingNumber(vector<int>& nums) {
           int n = nums.size();
           for(int i = 0; i < nums.size(); i++){
               while(nums[i] < n && nums[i] != i)
                   swap(nums[i], nums[nums[i]]);
           }
           for(int i = 0; i < nums.size(); i++){
               if(nums[i] != i)
                   return i;
           }
           return n;
       }
   };
   ```

3. 不修改原数组

```
class Solution {
public:
    int missingNumber(vector<int>& nums) {
        int sum = 0;
        int aim = nums.size() * (nums.size() + 1) / 2;
        for(int i = 0; i < nums.size(); i++)
            sum += nums[i];
        return aim - sum;
    }
};
```

75. 颜色分类（中等）

1. 题目描述

   给定一个包含红色、白色和蓝色，一共 n 个元素的数组，原地对它们进行排序，使得相同颜色的元素相邻，并按照红色、白色、蓝色顺序排列。

   此题中，我们使用整数 0、 1 和 2 分别表示红色、白色和蓝色。

   注意: 不能使用代码库中的排序函数来解决这道题。

   ```
   示例:
   输入: [2,0,2,1,1,0]
   输出: [0,0,1,1,2,2]
   ```

   进阶:

   - 一个直观的解决方案是使用计数排序的两趟扫描算法。
   - 首先，迭代计算出0、1 和 2 元素的个数，然后按照0、1、2的排序，重写当前数组。
   - 你能想出一个仅使用常数空间的一趟扫描算法吗?

2. 简单实现

   仿照三路快排

```
class Solution {
public:
    void sortColors(vector<int>& nums) {
        int l = 0, r = nums.size() - 1;
        int i = l;//0...l是0, l..i是1, i...r是待处理的, r右边的是2
        while(i <= r){
            while(i < r && nums[i] == 1) i++;
            if(i <= r){
                if(nums[i] == 2)
                    swap(nums[i], nums[r--]);
                else
                    swap(nums[l++], nums[i++]);
            }
        }
    }
};
```

490. 迷宫（中等）

要会员

37. 解数独（困难）

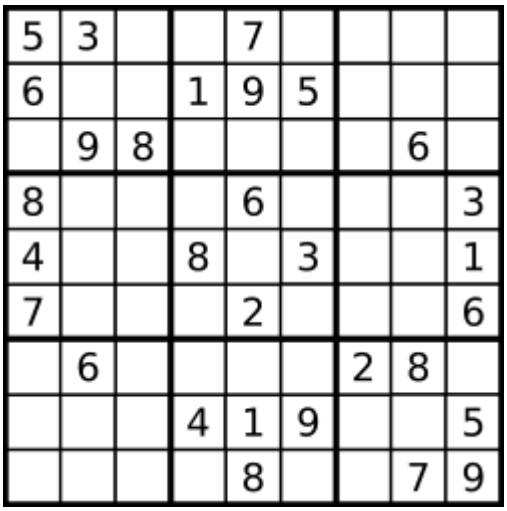
1. 题目描述

   编写一个程序，通过已填充的空格来解决数独问题。

   一个数独的解法需遵循如下规则：

   - 数字 1-9 在每一行只能出现一次
   - 数字 1-9 在每一列只能出现一次。
   - 数字 1-9 在每一个以粗实线分隔的 3x3 宫内只能出现一次。

   空白格用 '.' 表示。



5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

   一个数独。

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

   答案被标成红色。

   Note:

   - 给定的数独序列只包含数字 1-9 和字符 '.' 。
   - 你可以假设给定的数独只有唯一解。

- 给定数独永远是 9x9 形式的。

2. 简单实现

    1. 最基础想法：对各个空格子依次尝试可行的数

    可以改进的地方：先尝试可选择数量少的格子，可以减少搜索时间，但是实在没脑子写了。。。

```
class Solution {
public:
    unordered_map<int, unordered_set<int>> row;//记录各行已有数字
    unordered_map<int, unordered_set<int>> col;//记录各列已有数字
    unordered_map<int, unordered_set<int>> block;//记录各块已有数字
    bool done = false;
    bool judge(int x, int y, int n){//判断board[x][y]处能否放数字n
        return (row[x].find(n) == row[x].end()
                && col[y].find(n) == col[y].end()
                && block[x/3*3+y/3].find(n) == block[x/3*3+y/3].end());
    }
    void dfs(vector<vector<char>>& board, int place){//放置board[place/9]
[place%9]
        if(done) return;
        int x = place / 9;
        int y = place % 9;
        if(board[x][y] != '.'){//已经放置，放置下一个
            if(place == 80)//结束
                done = true;
            else
                dfs(board, place+1);
            return;
        }
        for(int k = 1; k <= 9; k++){//依次尝试
            if(judge(x,y,k)){//可以放置
                //放置
                board[x][y] = k + '0';
                row[x].insert(k);
                col[y].insert(k);
                block[x/3*3+y/3].insert(k);
                int i, j;
                if(place == 80){//结束
                    done = true;
                    return;
                }
                dfs(board, place+1);//放置下一个
                if(done) return;//结束
                //恢复
                board[x][y] = '.';
                row[x].erase(k);
                col[y].erase(k);
                block[x/3*3+y/3].erase(k);
            }
        }
    }
    void solveSudoku(vector<vector<char>>& board) {
        for(int i = 0; i < 9; i++)
```

```
            for(int j = 0; j < 9; j++){
                if(board[i][j] == '.')
                    continue;
                else{
                    row[i].insert(board[i][j] - '0');
                    col[j].insert(board[i][j] - '0');
                    block[i/3*3+j/3].insert(board[i][j] - '0');
                }
            }
        dfs(board, 0);
    }
};
```

2. 最优代码

主要是表示每行出现的数，用数组比集合好

```
class Solution {
public:

    //每一行（列 or 单元格）的九个数字是否使用过
    bool row[9][9], col[9][9], cell[3][3][9];

    bool dfs(int x, int y, vector<vector<char>>& b){
        if(y == 9) x ++, y = 0;
        if(x == 9) return true;
        if(b[x][y] == '.'){
            for(int i = 0; i < 9; i ++){
                if(!row[x][i] && !col[y][i] && !cell[x/3][y/3][i]){
                    row[x][i] = col[y][i] = cell[x/3][y/3][i] = true;
                    b[x][y] = '1' + i;
                    if(dfs(x, y + 1, b)) return true;
                    row[x][i] = col[y][i] = cell[x/3][y/3][i] = false;
                    b[x][y] = '.';
                }
            }
        }
        else return dfs(x, y + 1, b);
        return false;
    }

    void solveSudoku(vector<vector<char>>& board) {
        for(int i = 0; i < 9; i ++){
            for(int j = 0; j < 9; j ++){
                char c = board[i][j];
                if(c != '.')
                    row[i][c-'1'] = col[j][c-'1'] = cell[i/3][j/3][c-'1'] =
true;
            }
        }
        dfs(0, 0, board);
    }
};
```

122. 买卖股票的最佳时机II（简单）

1. 题目描述

给定一个数组，它的第 i 个元素是一支给定股票第 i 天的价格。

设计一个算法来计算你所能获取的最大利润。你可以尽可能地完成更多的交易（多次买卖一支股票）。

注意：你不能同时参与多笔交易（你必须在再次购买前出售掉之前的股票）。

```
示例 1:
输入: [7,1,5,3,6,4]
输出: 7
解释: 在第 2 天（股票价格 = 1）的时候买入，在第 3 天（股票价格 = 5）的时候卖出，这笔交易所能获得利润 = 5-1 = 4 。
     随后，在第 4 天（股票价格 = 3）的时候买入，在第 5 天（股票价格 = 6）的时候卖出，这笔交易所能获得利润 = 6-3 = 3 。

示例 2:
输入: [1,2,3,4,5]
输出: 4
解释: 在第 1 天（股票价格 = 1）的时候买入，在第 5 天 （股票价格 = 5）的时候卖出，这笔交易所能获得利润 = 5-1 = 4 。
     注意你不能在第 1 天和第 2 天接连购买股票，之后再将它们卖出。
     因为这样属于同时参与了多笔交易，你必须在再次购买前出售掉之前的股票。

示例 3:
输入: [7,6,4,3,1]
输出: 0
解释: 在这种情况下，没有交易完成，所以最大利润为 0。
```

提示：

- 1 <= prices.length <= 3 * 10 ^ 4
- 0 <= prices[i] <= 10 ^ 4

2. 简单实现

```cpp
class Solution {
public:
    int maxProfit(vector<int>& prices) {
        int n = prices.size();
        if(n <= 1) return 0;
        vector<vector<int>> dp(n, vector<int>(2));//某一天不持有/持有股票的最大利润
        dp[0][0] = 0;
        dp[0][1] = -prices[0];
        for(int i = 1; i < n; i++){
            dp[i][0] = max(dp[i-1][0], dp[i-1][1] + prices[i]);
            dp[i][1] = max(dp[i-1][0] - prices[i], dp[i-1][1]);
        }
        return dp[n-1][0];
    }
};
```

3. 贪心遍历

```
class Solution {
public:
    int maxProfit(vector<int>& prices) {
       if(prices.empty())return 0;
        int len = prices.size();
        int i=1,min = prices[0],max = prices[0],res=0;
        for(int i=1;i<len;i++){
            if(prices[i]<max){
                res+=max-min;
                min = prices[i];
                max = prices[i];
            }
            else if(prices[i]>=max)
                max = prices[i];
        }
        res+=max-min;
        return res;
    }
};
```

767. 重构字符串（中等）

1. 题目描述

给定一个字符串S，检查是否能重新排布其中的字母，使得两相邻的字符不同。

若可行，输出任意可行的结果。若不可行，返回空字符串。

```
示例 1:
输入: S = "aab"
输出: "aba"

示例 2:
输入: S = "aaab"
输出: ""
```

注意: S 只包含小写字母并且长度在[1, 500]区间内。

2. 简单实现

```
class Solution {
public:
    string reorganizeString(string S) {
        int len = S.size();
        unordered_map<char, int> cnts;//计数
        for(auto c : S){
            if(++cnts[c] > (len + 1) / 2)//超过一半，肯定不行
                return "";
        }
        priority_queue<pair<int, char>> q;//大顶堆
```

```
        for(auto m : cnts)
            q.push({m.second, m.first});
        string ans(len, ' ');
        int i = 0;//先放偶数位
        while(!q.empty()){
            char c = q.top().second;
            int cnt = q.top().first;
            q.pop();
            while(cnt--){
                i = i >= len ? 1 : i;//到头了，开始放奇数位
                ans[i] = c;
                i = i + 2;//间隔放置
            }
        }
        return ans;
    }
};
```

110. 平衡二叉树（简单）

1. 题目描述

   给定一个二叉树，判断它是否是高度平衡的二叉树。

   本题中，一棵高度平衡二叉树定义为：一个二叉树每个节点 的左右两个子树的高度差的绝对值不超过1。

```
示例 1:
给定二叉树 [3,9,20,null,null,15,7]
  3
 / \
  9  20
    /  \
   15   7
返回 true 。
示例 2:
给定二叉树 [1,2,2,3,3,null,null,4,4]
    1
  / \
 2   2
/ \
3   3
/ \
4   4
返回 false 。
```

2. 简单实现

```
class Solution {
public:
    int balancedHeight(TreeNode* root){
        if(!root) return 0;//空树高度为0
        int lefth = balancedHeight(root->left);
```

```
        int righth = balancedHeight(root->right);
        if(lefth < 0 || righth < 0) return -1;//不是高度平衡
        if(abs(lefth - righth) > 1) return -1;//不是高度平衡
        return max(lefth, righth) + 1;//是高度平衡，返回树高
    }
    bool isBalanced(TreeNode* root) {
        if(balancedHeight(root) >= 0)
            return true;
        else
            return false;
    }
};
```

354. 俄罗斯套娃信封问题（困难）

1. 题目描述

   给定一些标记了宽度和高度的信封，宽度和高度以整数对形式 (w, h) 出现。当另一个信封的宽度和高度都比这个信封大的时候，这个信封就可以放进另一个信封里，如同俄罗斯套娃一样。

   请计算最多能有多少个信封能组成一组“俄罗斯套娃”信封（即可以把一个信封放到另一个信封里面）。

   说明: 不允许旋转信封。

```
示例:
输入: envelopes = [[5,4],[6,4],[6,7],[2,3]]
输出: 3
解释: 最多信封的个数为 3, 组合为: [2,3] => [5,4] => [6,7]。
```

2. 简单实现

   面试题 08.13. 堆箱子的变式

```
class Solution {
public:
    static bool cmp(vector<int>& a, vector<int>& b){//按宽度升序排序
        return a[0] < b[0];
    }
    int maxEnvelopes(vector<vector<int>>& envelopes) {
        if(envelopes.size() <= 1) return envelopes.size();
        sort(envelopes.begin(), envelopes.end(), cmp);//排序
        vector<int> dp(envelopes.size());//dp[i]表示以envelopes[i]为最外层的最多套娃数
        for(int i = 0; i < envelopes.size(); i++){
            dp[i] = 1;
            for(int j = i-1; j >= 0; j--){//可能可以套在当前envelope里面的都在前面
                if(envelopes[j][0] < envelopes[i][0] && envelopes[j][1] <
envelopes[i][1]){
                    dp[i] = max(dp[i], dp[j] + 1);
                    break;
                }
            }
            //插入排序，使dp升序
            int idx = i;
```

```
                while(idx >= 1 && dp[idx] < dp[idx-1]){
                    swap(dp[idx], dp[idx-1]);
                    swap(envelopes[idx], envelopes[idx-1]);
                    idx--;
                }
            }
            return dp.back();
        }
    };
```

3. 最优解法

对于二维的，有一种特别的解法——二维最长上升子序列，以前也遇到过，一旦要求严格递增，就应该想到该方法

方法：排序 + 最长递增子序列

该问题为最长递增子序列的二维问题。

我们要找到最长的序列，且满足 `seq[i+1]` 中的元素大于 `seq[i]` 中的元素。

该问题是输入是按任意顺序排列的——我们不能直接套用标准的 LIS 算法，需要先对数据进行排序。我们如何对数据进行排序，以便我们的 LIS 算法总能找到最佳答案？

我们可以在这里找到最长递增子序列的解决方法。如果您不熟悉该算法，请先理解该算法，因为它是解决此问题的前提条件。

算法：

假设我们知道了信封套娃顺序，那么从里向外的顺序必须是按 `w` 升序排序的子序列。

在对信封按 `w` 进行排序以后，我们可以找到 `h` 上最长递增子序列的长度。、

我们考虑输入 `[[1，3]，[1，4]，[1，5]，[2，3]]`，如果我们直接对 `h` 进行 LIS 算法，我们将会得到 `[3，4，5]`，显然这不是我们想要的答案，因为 `w` 相同的信封是不能够套娃的。

为了解决这个问题。我们可以按 `w` 进行升序排序，若 `w` 相同则按 `h` 降序排序。则上述输入排序后为 `[[1，5]，[1，4]，[1，3]，[2，3]]`，再对 `h` 进行 LIS 算法可以得到 `[5]`，长度为 1，是正确的答案。这个例子可能不明显。

我们将输入改为 `[[1，5]，[1，4]，[1，2]，[2，3]]`。则提取 `h` 为 `[5，4，2，3]`。我们对 `h` 进行 LIS 算法将得到 `[2，3]`，是正确的答案。

---

235. 二叉搜索树的最近公共祖先（简单）

1. 题目描述

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。
```

说明:

- 所有节点的值都是唯一的。
- p、q 为不同节点且均存在于给定的二叉搜索树中。

2. 简单实现

```
class Solution {
public:
    TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
        if(!root) return NULL;
        TreeNode* cur = root;
        if(p->val > q->val) swap(p, q);
        while(cur){
            if(cur->val >= p->val && cur->val <= q->val) return cur;
            else if(cur->val > q->val) cur = cur->left;
            else cur = cur->right;
        }
        return NULL;
    }
};
```

560. 和为K的子数组（中等）

1. 题目描述

给定一个整数数组和一个整数 k，你需要找到该数组中和为 k 的连续的子数组的个数。

```
示例 1 :
输入:nums = [1,1,1], k = 2
输出: 2 , [1,1] 与 [1,1] 为两种不同的情况。
```

说明 :

- 数组的长度为 [1, 20,000]。
- 数组中元素的范围是 [-1000, 1000] ，且整数 k 的范围是 [-1e7, 1e7]。

2. 最优解法

官方给的O(n^2)解法说能过，在c++里没过，看题解的O(n)方法，和以前做过的一道找数组中和为k的一对数的思想类似

这种方法背后的想法如下：如果累积总和（由 $sum[i]$ 表示加到 i^{th} 的和）最多两个指数是相同的，那么这些元素之间的元素总和为零。进一步扩展相同的想法，如果累计总和，在索引 i 和 j 处相差 k，即 $sum[i] - sum[j] = k$，则位于索引 i 和 j 之间的元素之和是 k。

基于这些想法，可以使用了一个哈希表 map，它用于存储所有可能的索引的累积总和以及相同累加和发生的次数。我们以以下形式存储数据：(sum_i，sum_i 的出现次数)。我们遍历数组$nums$并继续寻找累积总和。每当我们遇到一个新的和时，我们在hashmap中创建一个与该总和相对应的新条目。如果再次出现相同的和，我们增加与map中的和相对应的计数。此外，对于遇到的每个总和，我们还确定已经发生 $sum - k$ 总和的次数，因为它将确定具有总和 k 的子阵列发生到当前索引的次数。我们将 $count$ 增加相同的量。

```
class Solution {
public:
    int subarraySum(vector<int>& nums, int k) {
        int ans = 0, sum = 0;
        unordered_map<int, int> m;//当前前缀和为first的索引数为second
        m[0] = 1;
        for (int i = 0; i < nums.size(); i++) {
            sum += nums[i];
            if (m.find(sum - k) != m.end())//与前面的m[sum-k]个索引之间的元素和为k
                ans += m[sum - k];
            m[sum]++;
        }
        return ans;
    }
};
```

659. 分割数组为连续子序列（中等）

1. 题目描述

输入一个按升序排序的整数数组（可能包含重复数字），你需要将它们分割成几个子序列，其中每个子序列至少包含三个连续整数。返回你是否能做出这样的分割?

```
示例 1:
输入: [1,2,3,3,4,5]
输出: True
解释:
你可以分割出这样两个连续子序列 :
1, 2, 3
3, 4, 5

示例 2:
输入: [1,2,3,3,4,4,5,5]
输出: True
解释:
你可以分割出这样两个连续子序列 :
1, 2, 3, 4, 5
3, 4, 5
```

```
示例 3:
输入: [1,2,3,4,4,5]
输出: False
```

提示：输入的数组长度范围为 [1, 10000]

2. 正确解法——贪心

我们可以把问题想象为在一条数字直线上画区间。这让我们想到开始结束事件。

为了说明这个概念，我们假设有 `nums = [10, 10, 11, 11, 11, 11, 12, 12, 12, 12, 13]` ，没有 `9` 和 `14` 。我们必须有 2 个序列从 10 开始， 2 个序列从 11 开始， 3 个序列到 12 结束。

总的来说，当考虑一连串连续的整数 `x` ，令 `C = count[x+1] - count[x]` ，如果 `C > 0` ，必须有 `C` 个子序列从 `x+1` 开始，如果 `C < 0` ，必须有 `-C` 个子序列在 `x` 结束。即使区间内有更多的结束端点， `C` 至少是一个下界。

在上面的例子中， `count[11] - count[10] = 2` 和 `count[13] - count[12] = -3` 表明有两个子序列从 `11` 开始，且有三个子序列在 `12` 结束。

方法 2：贪心 [Accepted]

想法

我们把 3 个或更多的连续数字称作 *chain*。

我们从左到右考虑每一个数字 `x` ，如果 `x` 可以被添加到当前的 chain 中，我们将 `x` 添加到 chain 中，这一定会比创建一个新的 chain 要更好。

为什么呢？如果我们以 `x` 为起点新创建一个 chain ，这条新创建更短的链是可以接在之前的链上的，这可能会帮助我们避免创建一个从 `x` 开始的长度为 1 或者 2 的短链。

算法

我们将每个数字的出现次数统计好，记 `tails[x]` 是恰好在 `x` 之前结束的链的数目。

现在我们逐一考虑每个数字，如果有一个链恰好在 `x` 之前结束，我们将 `x` 加入此链中。否则，如果我们可以新建立一条链就新建。

我们可以优化额外空间到 $O(1)$，因为我们可以像 *方法 1* 一样统计数字的出现次数，而且我们只需要知道最后 3 个数字的出现次数即可。

```
class Solution {
public:
    bool isPossible(vector<int>& nums) {
        unordered_map<int, int> cnt;
        unordered_map<int, int> tail;
        for(int i = 0; i < nums.size(); i++)
            cnt[nums[i]]++;
        for(int i = 0; i < nums.size(); i++){
```

```
            if(cnt[nums[i]] == 0) continue;//已经被用完了
            else if(tail[nums[i]] > 0){//可以附加在某个链后
                tail[nums[i]]--;
                tail[nums[i]+1]++;
            }
            else if(cnt[nums[i]+1] > 0 && cnt[nums[i]+2] > 0){//必须新开一个链了
                cnt[nums[i]+1]--;
                cnt[nums[i]+2]--;
                tail[nums[i]+3]++;
            }
            else
                return false;
            cnt[nums[i]]--;
        }
        return true;
    }
};
```

16. 最接近的三数之和（中等）

1. 题目描述

   给定一个包括 n 个整数的数组 nums 和 一个目标值 target。找出 nums 中的三个整数，使得它们的和与 target 最接近。返回这三个数的和。假定每组输入只存在唯一答案。

```
例如，给定数组 nums = [-1, 2, 1, -4], 和 target = 1.
与 target 最接近的三个数的和为 2. (-1 + 2 + 1 = 2).
```

2. 简单实现

   排序+遍历+滑窗，O(n^2)

```
class Solution {
public:
    int threeSumClosest(vector<int>& nums, int target) {
        sort(nums.begin(), nums.end());
        int n = nums.size();
        int gap = INT_MAX;
        int ans = -1;
        for(int i = 0; i < n - 2; i++){
            int l = i+1, r = n-1;
            while(l < r){
                int sum = nums[i] + nums[l] + nums[r];
                int diff = sum - target;
                if(abs(diff) < gap){
                    gap = abs(diff);
                    ans = sum;
                    if(ans == target)
                        return ans;
                }
                if(diff > 0)
                    r--;
```

```
                else
                    l++;
            }
        }
        return ans;
    }
};
```

126. 单词接龙II（困难）

1. 题目描述

   给定两个单词（beginWord 和 endWord）和一个字典 wordList，找出所有从 beginWord 到 endWord 的最短转换序列。转换需遵循如下规则：

   - 每次转换只能改变一个字母。
   - 转换过程中的中间单词必须是字典中的单词。

   说明:

   - 如果不存在这样的转换序列，返回一个空列表。
   - 所有单词具有相同的长度。
   - 所有单词只由小写字母组成。
   - 字典中不存在重复的单词。
   - 你可以假设 beginWord 和 endWord 是非空的，且二者不相同。

```
示例 1:
输入:
beginWord = "hit",
endWord = "cog",
wordList = ["hot","dot","dog","lot","log","cog"]
输出:
[
  ["hit","hot","dot","dog","cog"],
  ["hit","hot","lot","log","cog"]
]

示例 2:
输入:
beginWord = "hit"
endWord = "cog"
wordList = ["hot","dot","dog","lot","log"]
输出: []
解释: endWord "cog" 不在字典中，所以不存在符合要求的转换序列。
```

2. 正确解法现

   难点在于单一的dfs会超时，要BFS构建前驱路径，再在dfs中恢复路径，具体见代码

```
class Solution {
    void dfs(unordered_map<string,unordered_set<string> > &trace, const string
&last, vector<string> path, vector<vector<string> > &vs){
        path.push_back(last);
```

```
        if(trace.count(last)==0){
            reverse(path.begin(),path.end());
            vs.push_back(path);
            return;
        }
        for(const string &word:trace[last])
            dfs(trace,word,path,vs);
    }
public:
    vector<vector<string>> findLadders(string begin, string end, vector<string>&
wordList) {
        unordered_set<string> dict(wordList.begin(),wordList.end());
        if (dict.count(end) == 0) return {};
        unordered_map<string, unordered_set<string> > trace;//所有可能的前驱
        unordered_set<string> q = {begin}, dels;
        for(; !q.empty() && trace.find(end) == trace.end(); q=dels){
            for(const string &word:q)
                dict.erase(word);
            dels.clear();
            for(const string &word:q){
                for(int i=0; i<word.length(); ++i){
                    string s=word;
                    for(char ch='a'; ch<='z'; ++ch){
                        if(word[i]==ch) continue;
                        s[i] = ch;
                        if(dict.find(s)==dict.end()) continue;
                        trace[s].insert(word);
                        dels.insert(s);
                    }
                }
            }
        }
        if(trace.size()==0) return {};
        vector<vector<string> > result;
        dfs(trace,end,{},result);
        return result;
    }
};
```

229. 求众数（中等）

1. 题目描述

   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]
```

2. 简单实现

实际上还是摩尔投票法，可以选出最多的几个数

如果至多选一个代表，那他的票数至少要超过一半（ ⌊ 1/2 ⌋ ）的票数；

如果至多选两个代表，那他们的票数至少要超过 ⌊ 1/3 ⌋ 的票数；

如果至多选m个代表，那他们的票数至少要超过 ⌊ 1/(m+1) ⌋ 的票数。

所以以后碰到这样的问题，而且要求达到线性的时间复杂度以及常量级的空间复杂度，直接套上摩尔投票法。

```cpp
class Solution {
public:
    vector<int> majorityElement(vector<int>& nums) {
        int size = nums.size();
        if(size <= 1) return nums;
        int num1, num2;
        int cnt1 = -1, cnt2 = -1;
        for(int i = 0; i < size; i++){
            if(cnt1 == -1 || (cnt1 == 0 && nums[i] != num2)){//新数
                num1 = nums[i];
                cnt1 = 1;
            }
            else if((cnt2 == -1 || cnt2 == 0) && nums[i] != num1){
                num2 = nums[i];
                cnt2 = 1;
            }
            else if(nums[i] == num1)//累计
                cnt1++;
            else if(nums[i] == num2)
                cnt2++;
            else{//抵消
                cnt1--;
                cnt2--;
            }
        }
        vector<int> ans;
        if(cnt1 > 0){//num1可以为候选人
            cnt1 = 0;
            for(int i = 0; i < size; i++)
                if(nums[i] == num1)
                    cnt1++;
        }
```

```
        if(cnt1 > size/3) ans.push_back(num1);//超过1/3
        if(cnt2 > 0){
            cnt2 = 0;
            for(int i = 0; i < size; i++)
                if(nums[i] == num2)
                    cnt2++;
        }
        if(cnt2 > size/3) ans.push_back(num2);
        return ans;
    }
};
```

175. 组合两个表（简单）

1. 题目描述

SQL架构：

```
Create table Person (PersonId int, FirstName varchar(255), LastName varchar(255))
Create table Address (AddressId int, PersonId int, City varchar(255), State varchar(255))
Truncate table Person
insert into Person (PersonId, LastName, FirstName) values ('1', 'Wang', 'Allen')
Truncate table Address
insert into Address (AddressId, PersonId, City, State) values ('1', '2', 'New York City', 'New York')
```

表1: `Person`

```
+-------------+---------+
| 列名         | 类型     |
+-------------+---------+
| PersonId    | int     |
| FirstName   | varchar |
| LastName    | varchar |
+-------------+---------+
PersonId 是上表主键
```

表2: `Address`

```
+-------------+---------+
| 列名         | 类型    |
+-------------+---------+
| AddressId   | int     |
| PersonId    | int     |
| City        | varchar |
| State       | varchar |
+-------------+---------+
AddressId 是上表主键
```

编写一个 SQL 查询，满足条件：无论 person 是否有地址信息，都需要基于上述两表提供 person 的以下信息：

```
FirstName, LastName, City, State
```

2. 简单实现

方法：使用 `outer join`

算法

因为表 **Address** 中的 **personId** 是表 **Person** 的外关键字，所以我们可以连接这两个表来获取一个人的地址信息。

考虑到可能不是每个人都有地址信息，我们应该使用 `outer join` 而不是默认的 `inner join`。

MySQL

```
select FirstName, LastName, City, State
from Person left join Address
on Person.PersonId = Address.PersonId
;
```

> 注意：如果没有某个人的地址信息，使用 `where` 子句过滤记录将失败，因为它不会显示姓名信息。

```
select FirstName, LastName, City, State
from Person left join Address
on Person.PersonId = Address.PersonId;
```

324. 摆动排序II（中等）

1. 题目描述

给定一个无序的数组 nums，将它重新排列成 nums[0] < nums[1] > nums[2] < nums[3]... 的顺序。

```
示例 1:
输入: nums = [1, 5, 1, 1, 6, 4]
输出: 一个可能的答案是 [1, 4, 1, 5, 1, 6]

示例 2:
输入: nums = [1, 3, 2, 2, 3, 1]
输出: 一个可能的答案是 [2, 3, 1, 3, 1, 2]
```

说明: 你可以假设所有输入都会得到有效的结果。

进阶: 你能用 O(n) 时间复杂度和 / 或原地 O(1) 额外空间来实现吗?

2. 简单实现

先排序再分开插是O(nlogn)的复杂度

考虑到不需要严格排序，只需要将数组分为大小的两部分，因此可以需要先线性时间选择找到中位数，然后三路partition分成三部分，就可以接一个插一个了，整体复杂度O(n)

```
class Solution {
public:
    void wiggleSort(vector<int>& nums) {
        int len = nums.size();
        quickSelect(nums, 0, len, len / 2);
        auto midptr = nums.begin() + len / 2;
        int mid = *midptr;

        // 3-way-partition
        int i = 0, j = 0, k = nums.size() - 1;
        while(j < k){
            if(nums[j] > mid){
                swap(nums[j], nums[k]);
                --k;
            }
            else if(nums[j] < mid){
                swap(nums[j], nums[i]);
                ++i;
                ++j;
            }
            else{
                ++j;
            }
        }

        if(nums.size() % 2) ++midptr;
        vector<int> tmp1(nums.begin(), midptr);
        vector<int> tmp2(midptr, nums.end());
        for(int i = 0; i < tmp1.size(); ++i){
            nums[2 * i] = tmp1[tmp1.size() - 1 - i];
        }
        for(int i = 0; i < tmp2.size(); ++i){
            nums[2 * i + 1] = tmp2[tmp2.size() - 1 - i];
        }
    }

private:
    void quickSelect(vector<int> &nums, int begin, int end, int n){
        int t = nums[end - 1];
        int i = begin, j = begin;
        while(j < end){
            if(nums[j] <= t){
                swap(nums[i++], nums[j++]);
            }
            else{
                ++j;
            }
        }
        if(i - 1 > n){
            quickSelect(nums, begin, i - 1, n);
```

```
        }
        else if(i <= n){
            quickSelect(nums, i, end, n);
        }
    }
};
```

96. 不同的二叉树（中等）

1. 题目描述

   给定一个整数 n，求以 1 ... n 为节点组成的二叉搜索树有多少种?

```
示例:
输入: 3
输出: 5
解释: 给定 n = 3, 一共有 5 种不同结构的二叉搜索树:
   1          3     3      2      1
    \        /     /      / \      \
     3     2     1      1   3      2
    /     /       \                 \
   2     1         2                 3
```

2. 简单实现——记忆化递归

   - 实际上相当于是给定中序遍历结果1...n，求有多少种对应的二叉树
   - 进而可以想到，相当于依次取数组内各节点为根节点，然后其左边的数组为左子树中序遍历的结果，右边同理，从而构成递归。
   - 对于长度为n的中序遍历结果1...n，以第i个节点为根节点的对应的二叉树共有`numTrees(i-1) * numTrees(n-i)`种
   - 进一步观察到，种类数只与长度n有关，与值无关，因此可以以n递归，且可以记忆化处理，优化效率
   - 也可以写成动态规划，懒得改了

```
class Solution {
public:
    unordered_map<int, int> m;//中序遍历长度为n的数组可以生成的二叉树数量
    int numTrees(int n) {
        if(n <= 1) return 1;//空或只有一个节点，结果唯一
        if(m.find(n) != m.end()) return m[n];//之前求过，直接用
        int ans = 0;
        for(int i = 1; i <= n; i++)//依次以各个节点为根节点
            ans += numTrees(i-1) * numTrees(n-i);//左*右
        m[n] = ans;//记忆化存储
        return ans;
    }
};
```

3. 数学演绎法

$$G(n) = \sum_{i=1}^{n} G(i-1) \cdot G(n-i)$$

事实上 $G(n)$函数的值被称为 卡塔兰数 C_n。卡塔兰数更便于计算的定义如下:

$$C_0 = 1, \qquad C_{n+1} = \frac{2(2n+1)}{n+2} C_n \tag{4}$$

证明过程可以参考上述文献，此处略去。

算法

有了公式 (4)， 计算 G_n/C_n 十分容易。以下为示例:

Java | Python

```
class Solution {
  public int numTrees(int n) {
    // Note: we should use long here instead of int, otherwise overflow
    long C = 1;
    for (int i = 0; i < n; ++i) {
      C = C * 2 * (2 * i + 1) / (i + 2);
    }
    return (int) C;
  }
}
```

复杂度分析

- 时间复杂度 : $O(N)$，只有一层循环。
- 空间复杂度 : $O(1)$，只需要一个变量来存储中间与最终结果。

375. 猜数字大小（中等）

1. 题目描述

我们正在玩一个猜数游戏，游戏规则如下：

我从 1 到 n 之间选择一个数字，你来猜我选了哪个数字。

每次你猜错了，我都会告诉你，我选的数字比你的大了或者小了。

然而，当你猜了数字 x 并且猜错了的时候，你需要支付金额为 x 的现金。直到你猜到我选的数字，你才算赢得了这个游戏。

```
示例:
n = 10, 我选择了8.
第一轮: 你猜我选择的数字是5，我会告诉你，我的数字更大一些，然后你需要支付5块。
第二轮: 你猜是7，我告诉你，我的数字更大一些，你支付7块。
第三轮: 你猜是9，我告诉你，我的数字更小一些，你支付9块。
游戏结束。8 就是我选的数字。
你最终要支付 5 + 7 + 9 = 21 块钱。
```

给定 n ≥ 1，计算你至少需要拥有多少现金才能确保你能赢得这个游戏。

2. 简单实现

区间dp的题，貌似我还不太熟悉的样子

首先，我们需要意识到我们在范围 $(1, n)$ 中猜数字的时候，需要考虑最坏情况下的代价。也就是说要算每次都猜错的情况下的总体最大开销。

在暴力算法中，我们首先在 $(1, n)$ 中任意挑选一个数字，假设它是个错误的猜测（最坏情况），我们需要用最小代价去猜到需要的数字。那么在一次尝试以后，答案要么在我们猜的数字的左边要么在右边，为了考虑最坏情况，我们需要考虑两者的较大值。因此，如果我们选择 i 作为第一次尝试，总体最小代价是：

$$\text{cost}(1, n) = i + \max\big(\text{cost}(1, i-1), \text{cost}(i+1, n)\big)$$

对于左右两段，我们分别考虑在段内选择一个数，并重复上面的过程来求得最小开销。

使用如上方法，我们能求得从 i 开始猜，猜到答案的最小代价。同样地，我们遍历 $(1, n)$ 中的所有数字并分别作为第一次尝试，求出每一个的代价，并输入最小值即为答案。

定义状态`dp[i][j]`表示在i...j中猜，至少需要多少现金才能确保赢，则状态转移方程为：

- `dp[i][j] = min(k + dp[i][k-1] + dp[k+1][j])`，其中，`i<=k<=j`，即依次以k为猜测点，分割猜测区

```
class Solution {
public:
    int getMoneyAmount(int n) {
        if(n == 1) return 0;
        if(n == 2) return 1;
        int dp[n+1][n+1];
        for(int i = 0; i <= n; i++){
            for(int j = 0; j <= n; j++){
                dp[i][j] = INT_MAX;
                if(i == j)
                    dp[i][j] = 0;//只有一个数，不需要花钱，初始化为0
            }
        }
        //按列来，从第2列开始
```

```
        for(int j = 2; j <= n; j++){
            //按行来，从下往上
            for(int i = j-1; i >= 1; i--){
                //算除了两端i,j的每一个分割点
                for(int k = i+1; k <= j-1; k++)
                    dp[i][j] = min(k+max(dp[i][j], dp[i][k-1],dp[k+1][j]));
                //算两端i,j
                dp[i][j]=min(dp[i][j],i+dp[i+1][j]);
                dp[i][j]=min(dp[i][j],j+dp[i][j-1]);
            }
        }
        return dp[1][n];
    }
};
```

616. 给字符串添加加粗标签（中等）

要会员

684. 冗余连接（中等）

1. 题目描述

在本问题中, 树指的是一个连通且无环的无向图。

输入一个图，该图由一个有着N个节点 (节点值不重复1, 2, ..., N) 的树及一条附加的边构成。附加的边的两个顶点包含在1到N中间，这条附加的边不属于树中已存在的边。

结果图是一个以边组成的二维数组。每一个边的元素是一对[u, v]，满足 u < v，表示连接顶点u 和v的无向图的边。

返回一条可以删去的边，使得结果图是一个有着N个节点的树。如果有多个答案，则返回二维数组中最后出现的边。答案边 [u, v] 应满足相同的格式 u < v。

```
示例 1:
输入: [[1,2], [1,3], [2,3]]
输出: [2,3]
解释: 给定的无向图为:
  1
 / \
2 - 3

示例 2:
输入: [[1,2], [2,3], [3,4], [1,4], [1,5]]
输出: [1,4]
解释: 给定的无向图为:
5 - 1 - 2
    |   |
    4 - 3
```

注意:

- 输入的二维数组大小在 3 到 1000。
- 二维数组中的整数在1到N之间，其中N是输入数组的大小。

2. 正确解法——并查集

   合并集合且以较小值为根，对新的边a,b若属于同一根，那么a,b已经可以通过根连通，再加上a-b边后，就会构成环，所以得删。从前向后遍历，出现环的地方就是答案吗

```
class Solution {
public:
    int find(int n, vector<int> &rp){
        int num = n;
        while(rp[num] != num)
            num = rp[num];
        return num;
    }
    vector<int> findRedundantConnection(vector<vector<int>>& edges) {
        int sz = edges.size();
        vector<int> rp(sz+1);
        for(int i=0;i<sz;i++)
            rp[i] = i;
        for(int j=0;j<sz;j++){
            // 找到边上两个节点所在集合的代表节点
            int set1 = find(edges[j][0], rp);
            int set2 = find(edges[j][1], rp);
            if(set1 == set2)  // 两个集合代表节点相同，说明出现环，返回答案
                return edges[j];
            else    // 两个集合独立，合并集合。
                rp[set2] = set1;
        }
        return {0, 0};
    }
};
```

727. 最小窗口序列（困难）

要会员

729. 我的日程安排表I（中等）

1. 题目描述

   实现一个 MyCalendar 类来存放你的日程安排。如果要添加的时间内没有其他安排，则可以存储这个新的日程安排。

   MyCalendar 有一个 book(int start, int end)方法。它意味着在 start 到 end 时间内增加一个日程安排，注意，这里的时间是半开区间，即 [start, end), 实数 x 的范围为， start <= x < end。

   当两个日程安排有一些时间上的交叉时（例如两个日程安排都在同一时间内），就会产生重复预订。

   每次调用 MyCalendar.book方法时，如果可以将日程安排成功添加到日历中而不会导致重复预订，返回 true。否则，返回 false 并且不要将该日程安排添加到日历中。

   请按照以下步骤调用 MyCalendar 类: MyCalendar cal = new MyCalendar(); MyCalendar.book(start, end)

```
示例 1:
MyCalendar();
MyCalendar.book(10, 20); // returns true
MyCalendar.book(15, 25); // returns false
MyCalendar.book(20, 30); // returns true
解释:
第一个日程安排可以添加到日历中.  第二个日程安排不能添加到日历中，因为时间 15 已经被第一个日程安排预定了。
第三个日程安排可以添加到日历中，因为第一个日程安排并不包含时间 20 。
```

说明:

- 每个测试用例，调用 MyCalendar.book 函数最多不超过 100次。
- 调用函数 MyCalendar.book(start, end)时， start 和 end 的取值范围为 [0, 10^9]。

2. 简单实现

只有100次调用，暴力判断即可，再多可以用map

```
class MyCalendar {
public:
    vector<pair<int, int>> data;
    MyCalendar() {
    }
    bool book(int start, int end) {
        for(int i = 0; i < data.size(); i++){
            if(start >= data[i].second || end <= data[i].first)
                continue;
            else
                return false;
        }
        data.push_back({start, end});
        return true;
    }
};
```

202. 快乐数（简单）

1. 题目描述

编写一个算法来判断一个数 n 是不是快乐数。

「快乐数」定义为：对于一个正整数，每一次将该数替换为它每个位置上的数字的平方和，然后重复这个过程直到这个数变为 1，也可能是 无限循环 但始终变不到 1。如果 可以变为 1，那么这个数就是快乐数。

如果 n 是快乐数就返回 True ； 不是，则返回 False 。

```
示例:
输入: 19
输出: true
解释:
12 + 92 = 82
82 + 22 = 68
62 + 82 = 100
12 + 02 + 02 = 1
```

2. 简单实现

```
class Solution {
public:
    bool isHappy(int n) {
        unordered_set<int> s;
        s.insert(n);
        while(1){
            int cur = 0;
            while(n>0){
                int tmp = n % 10;
                n /= 10;
                cur += tmp*tmp;
            }
            if(cur == 1)
                return true;
            if(s.find(cur) != s.end())
                return false;
            s.insert(cur);
            n = cur;
        }
        return false;
    }
};
```

771. 宝石与石头（简单）

1. 题目描述

给定字符串J 代表石头中宝石的类型，和字符串 S代表你拥有的石头。 S 中每个字符代表了一种你拥有的石头的类型，你想知道你拥有的石头中有多少是宝石。

J 中的字母不重复，J 和 S中的所有字符都是字母。字母区分大小写，因此"a"和"A"是不同类型的石头。

```
示例 1:
输入: J = "aA", S = "aAAbbbb"
输出: 3

示例 2:
输入: J = "z", S = "ZZ"
输出: 0
```

注意:

- S 和 J 最多含有50个字母。
- J 中的字符不重复。

2. 简单实现

```
class Solution {
public:
    int numJewelsInStones(string J, string S) {
        unordered_set<char> s;
        for(int i = 0; i < J.size(); i++)
            s.insert(J[i]);
        int ans = 0;
        for(int i = 0; i < S.size(); i++)
            if(s.find(S[i]) != s.end())
                ans++;
        return ans;
    }
};
```

276. 栅栏涂色（简单）

要会员

951. 翻转等价二叉树（中等）

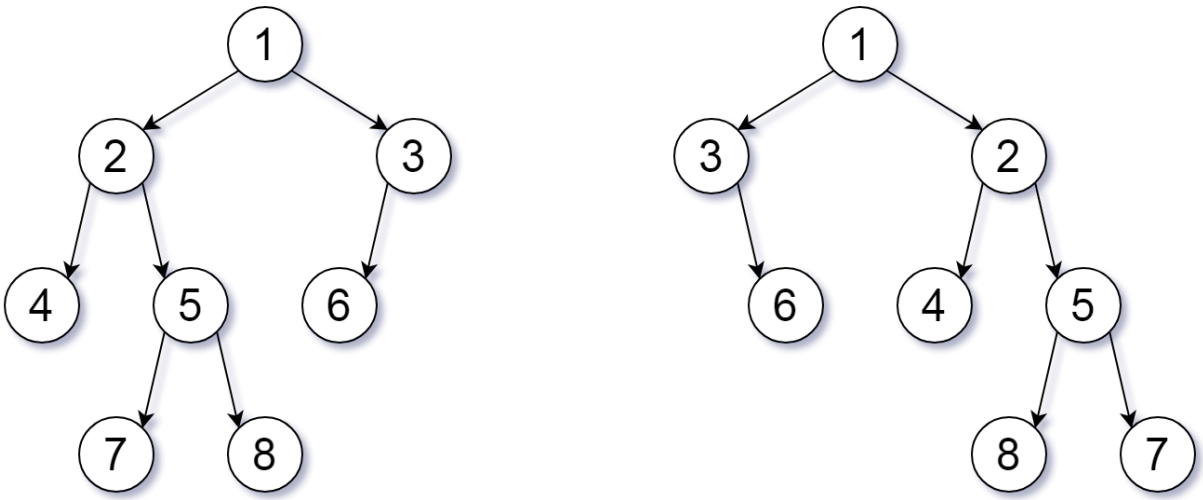
1. 题目描述

我们可以为二叉树 T 定义一个翻转操作，如下所示：选择任意节点，然后交换它的左子树和右子树。

只要经过一定次数的翻转操作后，能使 X 等于 Y，我们就称二叉树 X 翻转等价于二叉树 Y。

编写一个判断两个二叉树是否是翻转等价的函数。这些树由根节点 root1 和 root2 给出。

示例：



```
输入：root1 = [1,2,3,4,5,6,null,null,null,7,8], root2 =
[1,3,2,null,6,4,5,null,null,null,null,8,7]
输出：true
解释：我们翻转值为 1，3 以及 5 的三个节点。
```

提示：

- 每棵树最多有 100 个节点。
- 每棵树中的每个值都是唯一的、在 [0, 99] 范围内的整数。

2. 简单实现

```
class Solution {
public:
    bool flipEquiv(TreeNode* root1, TreeNode* root2) {
        if(!root1 && !root2) return true;//都空
        if(!root1 || !root2) return false;//有一个不空
        if(root1->val != root2->val) return false;//根节点值必须相等
        if(flipEquiv(root1->left, root2->left)&&flipEquiv(root1->right, root2-
>right))//不翻转
            return true;
        if(flipEquiv(root1->left, root2->right)&&flipEquiv(root1->right, root2-
>left))//翻转
            return true;
        return false;
    }
};
```

3. 思路二

方法二： 标准态遍历

思路

让树中所有节点的左孩子都小于右孩子，如果当前不满足就翻转。我们把这种状态的二叉树称为 `标准态` 。所有等价二叉树在转换成标准态后都是完全一样的。

算法

用深度优先遍历来对比这两棵树在标准态下是否完全一致。对于两颗等价树，在标准态下遍历的结果一定是一样的。

1088. 易混淆数（困难）

要会员

117. 填充每个节点的下一个右侧节点指针（中等）

1. 题目描述

给定一个二叉树

```
struct Node {
  int val;
  Node *left;
  Node *right;
  Node *next;
}
```

填充它的每个 next 指针，让这个指针指向其下一个右侧节点。如果找不到下一个右侧节点，则将 next 指针设置为 NULL。

初始状态下，所有 next 指针都被设置为 NULL。

进阶：

- 你只能使用常量级额外空间。
- 使用递归解题也符合要求，本题中递归程序占用的栈空间不算做额外的空间复杂度。

示例：

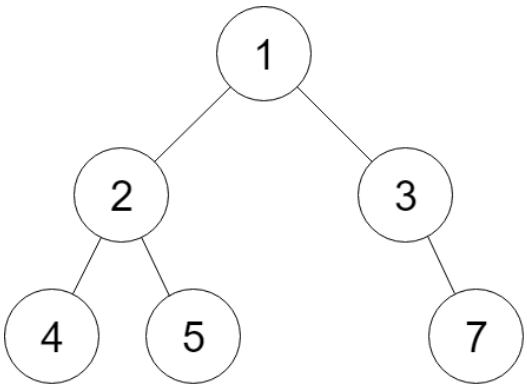


Figure A

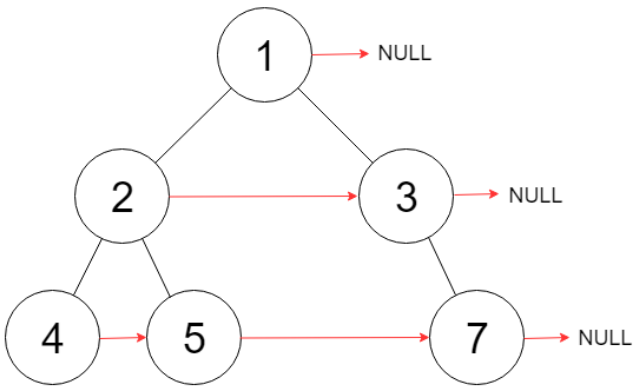


Figure B

```
输入：root = [1,2,3,4,5,null,7]
输出：[1,#,2,3,#,4,5,7,#]
解释：给定二叉树如图 A 所示，你的函数应该填充它的每个 next 指针，以指向其下一个右侧节点，如图 B 所示。
```

提示：

- 树中的节点数小于 6000
- -100 <= node.val <= 100

2. 简单实现

```
class Solution {
public:
    Node* connect(Node* root) {
        if (!root) return NULL;
        queue<Node*> q;
        q.push(root);
        while(!q.empty()){
            int size = q.size();
            for(int i = 0; i < size; i++){
                Node* temp = q.front();
                q.pop();
                if(temp->left) q.push(temp->left);
```

```
                if(temp->right) q.push(temp->right);
                if(i != size-1)
                    temp->next = q.front();
            }
        }
        return root;
    }
};
```

1048. 最长字符串链（中等）

1. 题目描述

   给出一个单词列表，其中每个单词都由小写英文字母组成。

   如果我们可以在 word1 的任何地方添加一个字母使其变成 word2，那么我们认为 word1 是 word2 的前身。例如，"abc" 是 "abac" 的前身。

   词链是单词 [word_1, word_2, ..., word_k] 组成的序列，k >= 1，其中 word_1 是 word_2 的前身，word_2 是 word_3 的前身，依此类推。

   从给定单词列表 words 中选择单词组成词链，返回词链的最长可能长度。

   ```
   示例:
   输入: ["a","b","ba","bca","bda","bdca"]
   输出: 4
   解释: 最长单词链之一为 "a","ba","bda","bdca"。
   ```

   提示:

   - 1 <= words.length <= 1000
   - 1 <= words[i].length <= 16
   - words[i] 仅由小写英文字母组成。

2. 简单实现

   先把单词按长度排序，再从后向前遍历，如果 `words[i]->words[j]`，则 `cnt[i]=max(cnt[i], cnt[j]+1)`

```
class Solution {
public:
    static bool cmp(string& a, string& b){//排序函数
        return a.size() < b.size();
    }
    bool judge(string& a, string& b){//判断a是否为b的前身
        int i = 0, j = 0, k = 0;
        while (i < a.size() && j < b.size()) {
            if (a[i] != b[j]) {
                if (k == 1)
                    return false;
                k++;
            }
            else
                i++;
            j++;
```

```
        }
        return true;
    }
    int longestStrChain(vector<string>& words) {
        sort(words.begin(), words.end(), cmp);
        int size = words.size();
        vector<int> cnt(size, 1);
        int ans = 1;
        for(int i = size - 1; i >= 0; i--){//从后向前
            int len = words[i].size();
            for(int j = i + 1; j < size; j++){//向后找其后身
                if(len == words[j].size()) continue;
                if(len + 1 < words[j].size()) break;
                if(judge(words[i], words[j]))//找到一个
                    cnt[i] = max(cnt[i], cnt[j]+1);
            }
            ans = max(ans, cnt[i]);
        }
        return ans;
    }
};
```

231. 2的幂（简单）

1. 题目描述

   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: 20 = 1

示例 2:
输入: 16
输出: true
解释: 24 = 16

示例 3:
输入: 218
输出: false
```

2. 简单实现

```
class Solution {
public:
    bool isPowerOfTwo(int n) {
        if(n <= 0) return false;
        while((n & 1) == 0)//必须加括号，优先级问题
            n = n >> 1;
        return n == 1;
    }
};
```

3. 最优解法

   要记住啊啊啊啊

```
class Solution {
public:
    bool isPowerOfTwo(int n) {
        if(n <= 0) return false;
        return (n & (n-1)) == 0;
    }
};
```

286. 墙与门（中等）

要会员

389. 找不同（简单）

1. 题目描述

   给定两个字符串 s 和 t，它们只包含小写字母。

   字符串 t 由字符串 s 随机重排，然后在随机位置添加一个字母。

   请找出在 t 中被添加的字母。

```
示例:
输入:
s = "abcd"
t = "abcde"
输出:
e
解释:
'e' 是那个被添加的字母。
```

2. 简单实现

```
class Solution {
public:
    char findTheDifference(string s, string t) {
```

```
        sort(s.begin(), s.end());
        sort(t.begin(), t.end());
        int len = s.size();
        int i = 0;
        while(i < len){
            if(s[i] != t[i])
                return t[i];
            i++;
        }
        return t[len];
    }
};
```

3. ASCII码之差

```
class Solution {
public:
    char findTheDifference(string s, string t) {
        int ans = 0;
        for (char ch : t) ans += ch;
        for (char ch : s) ans -= ch;
        return (char)ans;
    }
};
```

415. 字符串相加（简单）

1. 题目描述

   给定两个字符串形式的非负整数 num1 和num2 ，计算它们的和。

   注意：

   - num1 和num2 的长度都小于 5100.
   - num1 和num2 都只包含数字 0-9.
   - num1 和num2 都不包含任何前导零。
   - 你不能使用任何内建 BigInteger 库， 也不能直接将输入的字符串转换为整数形式。

2. 简单实现

```
class Solution {
public:
    string addStrings(string num1, string num2) {
        string str;
        int cur = 0, i = num1.size()-1, j = num2.size()-1;
        while (i >= 0 || j >= 0 || cur != 0) {
            if (i >= 0) cur += num1[i--] - '0';
            if (j >= 0) cur += num2[j--] - '0';
            str += to_string (cur % 10);
            cur /= 10;
        }
        reverse(str.begin(), str.end());
        return str;
```

```
    }
};
```

803. 打砖块（困难）

1. 题目描述

   我们有一组包含1和0的网格；其中1表示砖块。 当且仅当一块砖直接连接到网格的顶部，或者它至少有一块相邻（4 个方向之一）砖块不会掉落时，它才不会落下。

   我们会依次消除一些砖块。每当我们消除 (i, j) 位置时， 对应位置的砖块（若存在）会消失，然后其他的砖块可能因为这个消除而落下。

   返回一个数组表示每次消除操作对应落下的砖块数目。

```
示例 1:
输入:
grid = [[1,0,0,0],[1,1,1,0]]
hits = [[1,0]]
输出: [2]
解释:
如果我们消除(1, 0)位置的砖块，在(1, 1) 和(1, 2) 的砖块会落下。所以我们应该返回2。

示例 2:
输入:
grid = [[1,0,0,0],[1,1,0,0]]
hits = [[1,1],[1,0]]
输出: [0,0]
解释:
当我们消除(1, 0)的砖块时，(1, 1)的砖块已经由于上一步消除而消失了。所以每次消除操作不会造成砖块落下。注意(1, 0)砖块不会记作落下的砖块。
```

   注意:

   - 网格的行数和列数的范围是[1, 200]。
   - 消除的数字不会超过网格的区域。
   - 可以保证每次的消除都不相同，并且位于网格的内部。
   - 一个消除的位置可能没有砖块，如果这样的话，就不会有砖块落下。

2. 正确解法

   唉差一点想到，一开始想歪了想用有向图23333

   题解带动画https://leetcode-cn.com/problems/bricks-falling-when-hit/solution/hen-you-qu-de-da-zhuan-kuai-by-muyids/

```
class Solution {
public:
    int dx[4] = {-1, 0, 1, 0}, dy[4] = {0, 1, 0, -1};
    vector<int> size;
    vector<int> f;

    vector<int> hitBricks(vector<vector<int>> &grid, vector<vector<int>> &hits) {
        int m = grid.size(), n = grid[0].size();
```

```
        int last = 0;
        const int N = m * n;
        vector<bool> st(N, false);
        vector<int> ans;
        for (int i = 0; i < m * n; i++) f.push_back(i);
        size = vector<int>(m * n, 1);
        vector<vector<int>> gst(grid);

        for (auto &v: hits) grid[v[0]][v[1]] = 0; // 初始化并查集
        for (int i = 0; i < N; i++) {
            if (grid[i / n][i % n]) {
                for (int d = 0; d < 4; d++) {
                    int x = dx[d] + i / n, y = dy[d] + i % n;
                    if (x >= 0 && x < m && y >= 0 && y < n && grid[x][y])
                        un(i, x * n + y);
                }
            }
        }

        for (int i = 0; i < n; i++) {
            if (grid[0][i]) {
                if (st[find(i)]) continue;
                st[find(i)] = true;
                last += size[find(i)];
            }
        }
        reverse(hits.begin(), hits.end());
        for (auto &h: hits) {
            if (gst[h[0]][h[1]] == 0) {
                ans.push_back(0);
                continue;
            }
            grid[h[0]][h[1]] = 1;
            for (int d = 0; d < 4; d++) {
                int x = h[0] + dx[d], y = h[1] + dy[d];
                if (x >= 0 && x < m && y >= 0 && y < n && grid[x][y])
                    un(h[0] * n + h[1], x * n + y);
            }

            int cnt = 0;
            st = vector<bool>(N, false);
            for (int i = 0; i < n; i++) {
                if (grid[0][i]) {
                    if (st[find(i)]) continue;
                    st[find(i)] = true;
                    cnt += size[find(i)];
                }
            }
            int cur = cnt - last - 1 == -1 ? 0 : cnt - last - 1;
            ans.push_back(cur);
            last = cnt;
        }
        reverse(ans.begin(), ans.end());
```

```
            return ans;
        }

        int find(int x) {
            if (x == f[x]) return x;
            return f[x] = find(f[x]);
        }

        void un(int x, int y) {
            int p = find(x), q = find(y);// 并
            if (p != q) {
                f[p] = q;
                size[q] += size[p];
            }
        }
    };
```

1231. 分享巧克力（困难）

要会员