346. Moving Average from Data Stream

Given a stream of integers and a window size, calculate the moving average of all integers in the sliding window. 给一个数据流和window size， 求window内的均值。给定一串数和sliding window的window size，要求每次O(1)计算mean和std

```
class MovingAverage {
public:
    /** Initialize your data structure here. */
    MovingAverage(int size) {
        this->size = size;
        sum = 0.0;
    }

    double next(int val) {
        if(size == 0) return 0.0;
        if(window.size() < size){
            window.push(val);
            sum += val;
        }
        else{
            sum = sum - window.front() + val;
            window.pop();
            window.push(val);
        }
        return sum/(double)window.size();
    }

private:
    int size;
    queue<int> window;
    double sum;
};

/**
 * Your MovingAverage object will be instantiated and called as such:
 * MovingAverage obj = new MovingAverage(size);
 * double param_1 = obj.next(val);
 */
```

follow up：要算标准差怎么做（利用Var[X]=(E[X]^2)-(E[X^2])）

641. Design Circular Deque

其实fix size queue也适用这个 or deque

```
class MyCircularDeque {
```

```
public:
    /** Initialize your data structure here. Set the size of the deque to be
k. */
    MyCircularDeque(int k) {
        queue = vector<int>(k, 0);
        count = 0;
        this->k = k;
        front = k-1;
        rear = 0;
    }

    /** Adds an item at the front of Deque. Return true if the operation is
successful. */
    bool insertFront(int value) {
        if(count == k) return false;
        queue[front] = value;
        front = (front - 1 + k) % k;
        count++;
        return true;
    }

    /** Adds an item at the rear of Deque. Return true if the operation is
successful. */
    bool insertLast(int value) {
        if(count == k) return false;
        queue[rear] = value;
        rear = (rear + 1) % k;
        count++;
        return true;
    }

    /** Deletes an item from the front of Deque. Return true if the
operation is successful. */
    bool deleteFront() {
        if(count == 0) return false;
        front = (front + 1) % k;
        count--;
        return true;
    }

    /** Deletes an item from the rear of Deque. Return true if the operation
is successful. */
    bool deleteLast() {
        if(count == 0) return false;
        rear = (rear -1 + k) % k;
        count--;
        return true;
    }
```

```
    /** Get the front item from the deque. */
    int getFront() {
        if(count == 0) return -1;
        return queue[(front + 1) % k];
    }

    /** Get the last item from the deque. */
    int getRear() {
        if(count == 0) return -1;
        return queue[(rear - 1 + k) % k];
    }

    /** Checks whether the circular deque is empty or not. */
    bool isEmpty() {
        return count == 0;
    }

    /** Checks whether the circular deque is full or not. */
    bool isFull() {
        return count == k;
    }

private:
    vector<int> queue;
    int k;
    int count;
    int front;
    int rear;
};
```

follow up: 多线程怎么做（只答了用一个mutex把整个数据结构锁起来）

47. Permutations II

contain duplicates

```
class Solution {
public:
    vector<vector<int>> permuteUnique(vector<int>& nums) {
        vector<vector<int>> result;
        set<vector<int>> ans;
        vector<bool> visited(nums.size(), false);
        backtracking(nums, ans, visited, vector<int>{});
        for(auto res: ans) result.push_back(res);
        return result;
    }

private:
    void backtracking(vector<int> &nums, set<vector<int>> &ans,
```

```
                        vector<bool> &visited, vector<int> cur){
        if(cur.size() == nums.size()){
            ans.insert(cur);
            return;
        }
        for(int i = 0; i < visited.size(); i++){
            if(visited[i] == false){
                visited[i] = true;
                cur.push_back(nums[i]);
                backtracking(nums, ans, visited, cur);
                cur.pop_back();
                visited[i] = false;
            }
        }
    }
};
```

315. Count of Smaller Numbers After Self

```
class Solution {
public:
    vector<int> countSmaller(vector<int>& nums) {
        if(nums.size() == 0) return {};
        vector<int> ans(nums.size(), 0);
        multiset<int> seen;
        seen.insert(nums.back());
        for(int i = nums.size()-2; i >= 0; i--){
            auto lo = seen.lower_bound(nums[i]);
            ans[i] = distance(seen.begin(), lo);
            seen.insert(nums[i]);
        }
        return ans;
    }
};
```

265. Paint House II

```
class Solution {
public:
    int minCostII(vector<vector<int>>& costs) {
        int n = costs.size();
        if(n == 0) return 0;
        int k = costs[0].size();
        if(k == 1){
            int ans = 0;
            for(auto c: costs) ans += c[0];
            return ans;
        }
```

```
        vector<int> dp(k, 0);
        int min1, min2;

        for(int i = 0; i < n; ++i){
            int min1_old = (i == 0)? 0: min1;
            int min2_old = (i == 0)? 0: min2;
            min1 = INT_MAX;
            min2 = INT_MAX;
            for(int j = 0; j < k; ++j){
                if(dp[j] != min1_old || min1_old == min2_old){
                    dp[j] = min1_old + costs[i][j];
                }
                else{ // min1_old occurred when painting house i-1 with
color j, so it cannot be added to dp[j]
                    dp[j] = min2_old + costs[i][j];
                }
                if(min1 <= dp[j]){
                    min2 = min(min2, dp[j]);
                }
                else{
                    min2 = min1;
                    min1 = dp[j];
                }
            }
        }
        return min1;
    }
};
```

431. Encode N-ary Tree to Binary Tree

```
/*
// Definition for a Node.
class Node {
public:
    int val = NULL;
    vector<Node*> children;

    Node() {}

    Node(int _val, vector<Node*> _children) {
        val = _val;
        children = _children;
    }
};
*/
/**
 * 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 an n-ary tree to a binary tree.
    TreeNode* encode(Node* root) {
        if(root == nullptr) return nullptr;
        TreeNode *ans = new TreeNode(root->val);
        if(root->children.size()) ans->left = encode(root->children[0]);
        TreeNode *cur = ans->left;
        for(int i = 1; i < root->children.size(); i++){
            cur->right = encode(root->children[i]);
            cur = cur->right;
        }
        return ans;
    }

    // Decodes your binary tree to an n-ary tree.
    Node* decode(TreeNode* root) {
        if(root == nullptr) return nullptr;
        Node *ans = new Node();
        ans->val = root->val;
        TreeNode *cur = root->left;
        while(cur != nullptr){
            ans->children.push_back(decode(cur));
            cur = cur->right;
        }
        return ans;
    }
};

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

follow up:任意给一个二叉树，可以还原成N-ary tree吗？如果不能，满足什么样条件的可以呢？？原题我是用hashmap+level-order traversal做的，然后follow up是不能，需要满足根节点没有右子树且其任意节点的右儿子深度不超过N，第二个条件我开始也没想出来，我可能表达的也不是很准确，其实就是root = root->right，然后直到null停下来，这个深度不能超过N，因为子节点数量有限。不能复制节点

146. LRU Cache

```
class LRUCache {
```

```
public:
    LRUCache(int capacity) {
        this->capacity = capacity;
    }

    int get(int key) {
        if(k2v.find(key) == k2v.end()) return -1;
        keypos.erase(k2p[key]);
        keypos.push_front(key);
        k2p[key] = keypos.begin();
        return k2v[key];
    }

    void put(int key, int value) {
        if(k2v.size() == capacity && k2v.find(key) == k2v.end()){
            k2p.erase(keypos.back());
            k2v.erase(keypos.back());
            keypos.pop_back();
        }
        else if(k2v.find(key) != k2v.end()){
            keypos.erase(k2p[key]);
        }
        keypos.push_front(key);
        k2p[key] = keypos.begin();
        k2v[key] = value;
    }

private:
    unordered_map<int, int> k2v;
    unordered_map<int, list<int>::iterator> k2p;
    list<int> keypos;
    int capacity;
};

/**
 * Your LRUCache object will be instantiated and called as such:
 * LRUCache obj = new LRUCache(capacity);
 * int param_1 = obj.get(key);
 * obj.put(key,value);
 */
```

O(1) both

44. Wildcard Matching

```
class Solution {
public:
    bool isMatch(string s, string p) {
```

```
        vector<vector<bool>> dp(s.size()+1, vector<bool>(p.size()+1,
false));
        dp[0][0] = true;
        for(int j = 0; j < p.size(); j++) if(p[j] == '*' && dp[0][j] ==
true) dp[0][j+1] = true;
        for(int i = 0; i < s.size(); i++){
            for(int j = 0; j < p.size(); j++){
                if(p[j] == s[i] || p[j] == '?') dp[i+1][j+1] = dp[i][j];
                else if(p[j] == '*') dp[i+1][j+1] = dp[i+1][j] || dp[i]
[j+1];
            }
        }
        return dp.back().back();
    }
};
```

56. Merge Intervals

```
/**
 * Definition for an interval.
 * struct Interval {
 *     int start;
 *     int end;
 *     Interval() : start(0), end(0) {}
 *     Interval(int s, int e) : start(s), end(e) {}
 * };
 */
class Solution {
public:
    vector<Interval> merge(vector<Interval>& intervals) {
        vector<Interval> result;
        auto cmp = [](Interval a, Interval b){return a.end < b.end;};
        sort(intervals.begin(), intervals.end(), cmp);
        while(intervals.size() > 1){
            Interval a = intervals.back(); intervals.pop_back();
            Interval b = intervals.back(); intervals.pop_back();
            if(a.start <= b.end){
                Interval c(min(b.start, a.start), a.end);
                intervals.push_back(c);
            }
            else{
                result.push_back(a);
                intervals.push_back(b);
            }
        }
        if(intervals.size()) result.push_back(intervals.back());
        return result;
    }
};
```

followup：给两个Array of intervals， 每个array内部是无序的，也可能有overlap。求这两个array所覆盖的区域之间有没有overlap

109. Convert Sorted List to Binary Search Tree

```
/**
 * 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* sortedListToBST(ListNode* head) {
        if(head == nullptr) return nullptr;
        if(head->next == nullptr) return new TreeNode(head->val);
        pair<ListNode *, ListNode *> spl = splite(head);
        ListNode *left = spl.first;
        ListNode *right = spl.second;
        TreeNode *root = new TreeNode(right->val);
        root->left = sortedListToBST(left);
        root->right = sortedListToBST(right->next);
        return root;
    }

private:
    pair<ListNode *, ListNode *> splite(ListNode *head){
        ListNode *dummy = new ListNode(-1);
        dummy->next = head;
        ListNode *slow = dummy;
        ListNode *fast = dummy;
        while(fast->next && fast->next->next){
            fast = fast->next->next;
            slow = slow->next;
        }
        ListNode *l1 = head;
        ListNode *l2 = slow->next;
```

```
        slow->next = nullptr;
        return pair<ListNode *, ListNode *>{l1, l2};
    }
};
```

把双向链表转化为二叉平衡数，写出了 O(n log n) 的算法，followup 问怎么优化到 O(n)，应该是先把二叉平衡数存入数组就可以了。一个双向链表和二叉搜索树互相转化，用递归，每次找到那个root节点然后对左右子树继续进行递归就好，不过最好先遍历一遍链表，然后用数组存，每次递归传入这个数组和对应的index范围，最后时间复杂度O(N)，空间复杂度也是，反向转化的话也是类似思路 sorted linked list to BST 双向链表里面存的非减序列 要求转为平衡二叉搜索树。要求： 自己写数据结构(树节点结构类似于双向链表 树左右 类似于node pre post) in-place 不能创建额外空间 可以用hashmap。follow up:反过来 树转链表 要求相同。给一个binary tree,如何在没有额外变长空间分配下把它按照in order的方式变成doubly linked list

200. Number of Islands

```
class Solution {
public:
    int numIslands(vector<vector<char>>& grid) {
        if(grid.size() == 0 || grid[0].size() == 0) return 0;
        int m = grid.size();
        int n = grid[0].size();
        vector<int> p(m*n, -2);
        for(int i = 0; i < m; i++){
            for(int j = 0; j < n; j++){
                if(grid[i][j] == '1'){
                    p[i*n+j] = -1;
                    if(i-1 >= 0 && p[(i-1)*n+j] != -2) u(p, i*n+j, (i-
1)*n+j);
                    if(j-1 >= 0 && p[i*n+j-1] != -2) u(p, i*n+j, i*n+j-1);
                }
            }
        }
        return count(p.begin(), p.end(), -1);
    }

private:
    int f(vector<int> &p, int x){
        if(p[x] == -1) return x;
        return f(p, p[x]);
    }

    void u(vector<int> &p, int x, int y){
        int xp = f(p, x);
        int yp = f(p, y);
        if(xp != yp) p[xp] = yp;
    }
};
```

694. Number of Distinct Islands

Count the number of **distinct** islands. An island is considered to be the same as another if and only if one island can be translated (and not rotated or reflected) to equal the other.

```
class Solution {
public:
    int numDistinctIslands(vector<vector<int>>& grid) {
        vector<vector<bool>> seen(grid.size(), vector<bool>(grid[0].size(),
false));
        set<set<int>> shapes;
        for(int r = 0; r < grid.size(); r++){
            for(int c = 0; c < grid[r].size(); c++){
                set<int> shape = {};
                dfs(grid, seen, shape, r, c, r, c);
                if(shape.empty() == false){
                    shapes.insert(shape);
                }
            }
        }
        return shapes.size();
    }

private:
    void dfs(vector<vector<int>>& grid, vector<vector<bool>> &seen, set<int>
&shape,
             int r, int c, int r0, int c0){
        if (0 <= r && r < grid.size() && 0 <= c && c < grid[0].size() &&
                grid[r][c] == 1 && !seen[r][c]) {
            seen[r][c] = true;
            shape.insert((r - r0) * 2 * grid[0].size() + (c - c0));
            dfs(grid, seen, shape, r+1, c, r0, c0);
            dfs(grid, seen, shape, r-1, c, r0, c0);
            dfs(grid, seen, shape, r, c+1, r0, c0);
            dfs(grid, seen, shape, r, c-1, r0, c0);
        }
    }
};
```

305. Number of Islands II

A 2d grid map of `m` rows and `n` columns is initially filled with water. We may perform an *addLand* operation which turns the water at position (row, col) into a land. Given a list of positions to operate, **count the number of islands after each addLandoperation**.

```
class Solution {
public:
    vector<int> numIslands2(int m, int n, vector<pair<int, int>>& positions)
{
```

```
        map<pair<int, int>, pair<int, int>> p;
        vector<int> ans;
        for(auto pos: positions){
            p[pos] = {-1, -1};
            int x = pos.first;
            int y = pos.second;
            if(x-1 >= 0 && p.find({x-1, y}) != p.end()) u(p, pos, {x-1, y});
            if(x+1 < m && p.find({x+1, y}) != p.end()) u(p, pos, {x+1, y});
            if(y-1 >= 0 && p.find({x, y-1}) != p.end()) u(p, pos, {x, y-1});
            if(y+1 < n && p.find({x, y+1}) != p.end()) u(p, pos, {x, y+1});
            int a = 0;
            for(auto v: p) if(v.second == pair<int, int>{-1, -1}) a++;
            ans.push_back(a);
        }
        return ans;
    }

private:
    void u(map<pair<int, int>, pair<int, int>> &p, pair<int, int> x,
pair<int, int> y){
        auto xp = f(p, x);
        auto yp = f(p, y);
        if(xp != yp) p[xp] = yp;
    }

    pair<int, int> f(map<pair<int, int>, pair<int, int>> &p, pair<int ,int>
x){
        if(p[x] == pair<int, int>{-1, -1}) return x;
        return f(p, p[x]);
    }
};
```

20. Valid Parentheses

仅有小括号，先用的stack，后面问如何优化O space

每次读入一个左括号或右括，判断当前字符串是否合法，答完用栈的方法之后的变种：要求O(1) extra memory，双向插入括号

```
class Solution {
public:
    bool isValid(string s) {
        map<char, char> mapping;
        mapping[')'] = '(';
        mapping[']'] = '[';
        mapping['}'] = '{';
        stack<char> stk;
        //stk.push('$');
```

```
        for(int i = 0; i < s.size(); i++){
            char cur = s.at(i);
            if(stk.empty() == false && stk.top() == mapping[cur]){
                stk.pop();
            }
            else{
                stk.push(cur);
            }
        }

        return stk.empty();//stk.top() == '$';
    }
};
```

76. Minimum Window Substring

Given a string S and a string T, find the minimum window in S which will contain all the characters in T in complexity O(n).

```
class Solution {
public:
    string minWindow(string s, string t) {
        if(s.size() == 0 || t.size() == 0) return "";
        int pos = 0;
        int len = INT_MAX;
        int i = 0, j = 0;
        unordered_map<char, int> ch2num;
        int cnt = t.size();
        for(auto c: t) ch2num[c]++;
        while(i <= j && j < s.size()){
            char a = s[j++];
            if(ch2num[a] > 0) cnt--;
            ch2num[a]--;
            while(cnt == 0){
                if(j-i < len) len = j-i, pos = i;
                char b = s[i++];
                if(ch2num[b] == 0) cnt++;
                ch2num[b]++;
            }
        }
        return len == INT_MAX? "": s.substr(pos, len);
    }
};
```

4. Median of Two Sorted Arrays

```
class Solution {
public:
    double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {
```

```
        const int n1 = nums1.size();
        const int n2 = nums2.size();
        if(n1 > n2) return findMedianSortedArrays(nums2, nums1);
        const int k = (n1 + n2 + 1) / 2;
        int l = 0;
        int r = n1;
        while(l < r){
            const int m1 = l + (r - l) / 2;
            const int m2 = k - m1;
            if(nums1[m1] < nums2[m2-1])
                l = m1 + 1;
            else r = m1;
        }
        const int m1 = l;
        const int m2 = k - l;
        const int c1 = max(m1 <= 0? INT_MIN: nums1[m1-1],
                           m2 <= 0? INT_MIN: nums2[m2-1]);
        if((n1 + n2) % 2 == 1) return c1;
        const int c2 = min(m1 >= n1? INT_MAX: nums1[m1],
                           m2 >= n2? INT_MAX: nums2[m2]);
        return (c1 + c2) / 2.0;
    }
};
```

773. Sliding Puzzle

```
class Solution {
public:
    int slidingPuzzle(vector<vector<int>>& board) {
        int ans = 0;
        string target = "123450";
        string start = "";
        for(auto row: board) for(auto e: row) start += string(1, '0'+e);
        queue<string> qstates;
        set<string> seen;
        qstates.push(start);
        seen.insert(start);
        while(qstates.empty() == false){
            int size = qstates.size();
            while(size--){
                string front = qstates.front(); qstates.pop();
                if(front == target) return ans;
                auto nextStates = getNextStates(front);
                for(auto state: nextStates){
                    if(seen.find(state) == seen.end()){
                        seen.insert(state);
                        qstates.push(state);
                    }
                }
```

```
                }
                ans++;
            }
            return -1;
        }

    private:
        vector<string> getNextStates(string cur){
            vector<int> dir = {0, 1, 0, -1, 0};
            vector<string> nexts;
            auto pos = cur.find('0');
            for (int k = 0; k < 4; ++k) {
                int ni = pos/3 + dir[k];
                int nj = pos%3 + dir[k+1];
                if (ni < 0 || nj < 0 || ni >= 2 || nj >= 3)
                    continue;
                swap(cur[pos], cur[ni*3+nj]);
                nexts.push_back(cur);
                swap(cur[pos], cur[ni*3+nj]);
            }
            return nexts;
        }
    };
```

改版华容道移动，一个4*4的棋盘，O代表空位，X或者Y代表棋子。每次可以移动一步，求问最少多少次可以移动至胜利。胜利的条件是，存在排成一列的或者一行或者一条斜线的X(或者Y)。

OXXY

XOOY

XXXY

YYOO

经典n-puzzle问题，但是面试的题目变成n*m的puzzle，follow-up是如何判断解是否存在。解法: A* algorithm, follow up可以在O(n*m)的时间根据奇偶性判断(https://www.cs.bham.ac.uk/~mdr/t ... lesSolvability.html)。我没有解出follow-up，但是面试官说没没关系只是要看思考逻辑。follow-up複雜度寫錯了，應該是O((nm)^2)或是O(nm lg(nm))才對

907. Sum of Subarray Minimums

最优解是O(N)，记录在每个position，左边和右边比它小的元素的个数

Given an array of integers `A`, find the sum of `min(B)`, where `B` ranges over every (contiguous) subarray of `A`.

Since the answer may be large, **return the answer modulo 10^9 + 7.**

```
class Solution {
public:
    int sumSubarrayMins(vector<int>& A) {
```

```
        int ans = 0;
        int n = A.size(), mod = 1e9+7;
        vector<int> left(n, 0);
        vector<int> right(n, 0);
        stack<pair<int, int>> s1, s2;
        for(int i = 0; i < n; i++){
            int count = 1;
            while(s1.empty() == false && s1.top().first > A[i]){
                count += s1.top().second;
                s1.pop();
            }
            s1.push({A[i], count});
            left[i] = count;
        }
        for(int i = n-1; i >= 0; i--){
            int count = 1;
            while(s2.empty() == false && s2.top().first >= A[i]){
                count += s2.top().second;
                s2.pop();
            }
            s2.push({A[i], count});
            right[i] = count;
        }
        for(int i = 0; i < n; i++)
            ans = (ans + A[i]*left[i]*right[i]) % mod;
        return ans;
    }
};
```

136. Single Number

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

如果出现两次的数总是相邻的，有没有比异或一遍更高效的做法。要把O(n)的复杂度继续优化，显然用Binary search，结合位置的奇偶性就可以了。

215. Kth Largest Element in an Array

```
class Solution {
public:
```

```
    int findKthLargest(vector<int>& nums, int k) {
        int i = 0, j = nums.size()-1;
        while(i <= j){
            int p = partition(nums, i, j);
            if(p == k-1) return nums[p];
            if(p > k-1) j = p-1;
            else i = p+1;
        }
        return -1;
    }

private:
    int partition(vector<int> &nums, int i, int j){
        int pval = nums[i];
        int ppos = i++;
        while(i <= j){
            if(nums[i] < pval && pval < nums[j]) swap(nums[i++], nums[j--]);
            if(nums[i] >= pval) i++;
            if(nums[j] <= pval) j--;
        }
        swap(nums[ppos], nums[j]);
        return j;
    }
};
```

不能用priority queue，必须用quick sort来写

53. Maximum Subarray

Given an integer array `nums`, find the contiguous subarray (containing at least one number) which has the largest sum and return its sum.

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

followup是找出两个subarray，使他们sum最大，这里我用的是两个数组保存每个位置左边的maximum subarray和右边的maximum subarray。然后找两个数组对应位置sum最大的就行了

99. Recover Binary Search Tree

Two elements of a binary search tree (BST) are swapped by mistake.

Recover the tree without changing its structure.

```
/**
 * 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 recoverTree(TreeNode* root) {
        dfs(root);
        swap(first->val, second->val);
    }

private:
    TreeNode *first = nullptr;
    TreeNode *second = nullptr;
    TreeNode *prev = new TreeNode(INT_MIN);

    void dfs(TreeNode *root){
        if(root == nullptr) return;
        dfs(root->left);
        if(first == nullptr && prev->val >= root->val) first = prev;
        if(first != nullptr && prev->val >= root->val)
            second = root;
        prev = root;
        dfs(root->right);
    }
};
```

72. Edit Distance

Given two words *word1* and *word2*, find the minimum number of operations required to convert *word1* to *word2*.

You have the following 3 operations permitted on a word:

1. Insert a character
2. Delete a character
3. Replace a character

```
class Solution {
```

```
public:
    int minDistance(string word1, string word2) {
        int l1 = word1.size(), l2 = word2.size();
        vector<vector<int>> dp(l1+1, vector<int>(l2+1, -1));
        for(int i = 0; i <= l2; i++) dp[0][i] = i;
        for(int i = 0; i <= l1; i++) dp[i][0] = i;
        for(int i = 1; i <= l1; i++){
            for(int j = 1; j <= l2; j++){
                if(word1[i-1] == word2[j-1]){
                    dp[i][j] = dp[i-1][j-1];
                }
                else{
                    dp[i][j] = min(dp[i-1][j-1],
                                   min(dp[i-1][j], dp[i][j-1]))+1;
                }
            }
        }
        return dp.back().back();
    }
};
```

69. Sqrt(x)

```
class Solution {
public:
    int mySqrt(int x) {
        int l = 1, r = x;
        while(l <= r){
            int m = l + (r - l) / 2;
            if(m == x / m) return m;
            else if(m < x / m) l = m+1;
            else r = m-1;
        }
        return r;
    }
};
```

935. Knight Dialer

给一个电话数字键盘，1-9的九宫格，然后拨号方式是马走日（下一个数字相对上一个数字的位置是个“日”字），问你拨n位共有多少种可能。

```
class Solution {
public:
    const unsigned int mod = 1000000000+7;
    const vector<vector<unsigned long long>> matrix = {
       //0, 1, 2, 3, 4, 5, 6, 7, 8, 9
        {0, 0, 0, 0, 1, 0, 1, 0, 0, 0}, // 0
        {0, 0, 0, 0, 0, 0, 1, 0, 1, 0}, // 1
```

```
        {0, 0, 0, 0, 0, 0, 0, 1, 0, 1}, // 2
        {0, 0, 0, 0, 1, 0, 0, 0, 1, 0}, // 3
        {1, 0, 0, 1, 0, 0, 0, 0, 0, 1}, // 4
        {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, // 5
        {1, 1, 0, 0, 0, 0, 0, 1, 0, 0}, // 6
        {0, 0, 1, 0, 0, 0, 1, 0, 0, 0}, // 7
        {0, 1, 0, 1, 0, 0, 0, 0, 0, 0}, // 8
        {0, 0, 1, 0, 1, 0, 0, 0, 0, 0}, // 9
    };
    int knightDialer(int N) {
        if(N == 1) return 10;
        auto vec = fasterPower(N-1);
        unsigned long long ans = 0;
        for(auto row: vec){
            for(auto e: row){
                cout<<e<<" ";
                ans = (ans + e) % mod;
            }
            cout<<endl;
        }
        return (int)ans;
    }

    vector<vector<unsigned long long>> fasterPower(int n){
        if(n == 1) return matrix;
        if(n % 2 == 1){
            auto half = fasterPower(n / 2);
            return multiply(multiply(half, half), matrix);
        }
        else{
            auto half = fasterPower(n / 2);
            return multiply(half, half);
        }
    }

    vector<vector<unsigned long long>> multiply(vector<vector<unsigned long
long>> a,
                                                vector<vector<unsigned long
long>> b){
        vector<vector<unsigned long long>> ans(10, vector<unsigned long
long>(10, 0));
        int i, j, k;
        for(int i = 0; i < 10; i++){
            for(int j = 0; j < 10; j++){
                ans[i][j] = 0;
                for(k = 0; k < 10; k++){
                    ans[i][j] = (ans[i][j] + (a[i][k]*b[k][j]) % mod) % mod;
                }
            }
```

```
        }

        return ans;
    }
};
```

50. Pow(x, n)

```
class Solution {
public:
    double myPow(double x, int n) {
        if(n == 0) return 1.0;
        if(x == 0.0) return 0.0;
        bool sign = n > 0? true: false;
        long p = labs((long)n);
        return sign? binaryPow(x, p): 1 / binaryPow(x, p);
    }

private:
    double binaryPow(double x, long n){
        if(n == 0) return 1.0;
        if(n % 2 == 1){
            double half = binaryPow(x, n/2);
            return half * half * x;
        }
        else{
            double half = binaryPow(x, n/2);
            return half*half;
        }
    }
};
```

follow up log(n)

28. Implement strStr()

```
class Solution {
public:
    int strStr(string haystack, string needle) {
        if(needle.size() == 0) return 0;
        int m = needle.size();
        int n = haystack.size();
        vector<int> lps = buildlps(needle);
        int i = 0, j = 0;
        while(i < n){
            if(needle[j] == haystack[i]) j++, i++;
            if(j == m) return i-j;
            else if(i < n && needle[j] != haystack[i]){
                if(j != 0) j = lps[j-1];
```

```
                else i++;
            }
        }
        return -1;
    }

private:
    vector<int> buildlps(string pattern){
        int m = pattern.size();
        vector<int> lps(m, 0);
        int len = 0;
        int i = 1;
        while(i < m){
            if(pattern[i] == pattern[len]){
                len++, lps[i] = len, i++;
            }
            else{
                if(len != 0) len = lps[len-1];
                else lps[i] = 0, i++;
            }
        }
        return lps;
    }
};
```

- 小车从给定起点到终点运货，给定一个graph，每一条边有最大运货的限制，同时限制小车走过边的条数，问小车最大的运货量。因为函数接口给的是当前的node，所以我想的是用二分最大运货量，bfs看能不能在规定步数内走通，后续考了二分的细节，bfs减少while循环的优化 **BFS，走k边，对能到终点的取max**
- 判断两条线段是否相交,先求交点，然后判断交点范围可解，可以思考的是处理斜率为无限大的直线的时候如何简化代码.... https://segmentfault.com/a/1190000004457595
- 给m个数组，每个数组里有n个值，都是按照升序排好了。现在从每个数组里分别取1个值，并对这些值求和，这个和记为y。求前x个最小的y。 比如：给定数组 num1 = [1,2,3] num2 = [4,9,10] x=2，则最小的两个值是： 1+4 = 5 2+4 = 6 **n指针， 每次移一个指针，移最小的那个**
- 给出Union tree的定义：自己和所有child的value都一样。求一个tree里面有多少个union tree，比如以下这个就有6个： 1 1 2 1 1 2 2 除了顶点的1以外，以其他点为顶点的tree都符合Union tree **DFS**
- 合并n个BST。只让我写了一个TreeNode to array的函数。https://www.geeksforgeeks.org/merge-two-balanced-binary-search-trees/
- 有n个城市，每个城市之间可能有路径相连，如果有路径，这个路径会有一个weight，代表的是最大能够承受的车的重量，另外有不同重量的车，能走的路不能超过最大承重，每条路通过路费1元，预算k元，写函数求能够从s开到t的车的最大重量。follow up，如何优化 **BFS+DP**

  开一辆车，给定车的重量。给出一个起点和一个终点，不同的路用不同的最大承重量。你用n个硬币，每通过一条路就要支付一个硬币。求能从起点到终点的最短路径（每条路长度都是1）.

- 假设有很多张图片，图片之间有遮挡关系，找到最底层的图片 follow up1：假设所有的图片都是矩形，给你俯视图，如何找到图片之间的遮挡关系，提一下就好，不用写 follow up2：给出一个possible的从最底层到最外层的图片顺序，就是topological sorting，因为我当时时间还够就写了一下

  给一个pixel matrix，matrix可以理解为照相得到，不同颜色的pixel组成不同的广告牌，判断广告牌之间的遮挡关系 如下

  [1,1,1,1]

  [1,1,2,2]

  [1,1,2,3]

  不同的数字代表不同的颜色。例如此表示有

  [1,1,1,1]

  [1,1,1,1]

  [1,1,1,1]

  [2,2]

  [2,2]

  [3]

  3种广告牌，而1被2所遮挡，2被3所遮挡，所以在照片中的pixel分布为：

  [1,1,1,1]

  [1,1,2,2]

  [1,1,2,3]

  求距离最远的广告牌的颜色。

  给一个图像，上面有若干矩形，矩形颜色各不相同，可能有重叠。可以想象成一摞带颜色的矩形无规律放在桌面上，然后从上往下看。问可能有哪些颜色的矩形可能是在最底下一层的。注意是可能。 比如： BBP BWG RGG 这时候可能的颜色是，B P R G

- 假设从原点出发，可以往上左右三个方向走，不能经过已经走过的节点，一共走了n步，问总共有多少种路径。dp_i表示走n步的可能性。考虑在i-1步的情况，第i-1步是往上，那么接下来就有三种可能性，而且前面i-2步可以以任意方式到达。因此是dp[i-2]*3, i-1步剩下的情况，就是往左往右，对于下一步i都只有两种选择，因此(dp[i-1]-dp[i-2])*2,最后上面的相加就行。**DP，dp[i] = dp[i-2]*3 + (dp[i-1]-dp[i-2])*2**

- 一个排序后的长度为N的int list，e.g.[1,2,3,4,5,6,7,8]，改变其中k个数字的位置，e.g. [1,3,2,4,5,7,6,8] (k=2)。要求返回正确顺序的array，时间复杂度**小于**NlogN，k << N，不要求inplace。时间不超过O(N). 面试官提示下口述的做法：**扫描，发现异常点以后，就把这个点和之前的那个点拿出来，这样扫描一遍后就会得到两个数组，一个是已经升序的，另外一个是2k未排序的。klogk排序乱序短数组，然后merge 2 sorted list。**

- 面试官说现在给你一个字符串，你可以增加或者删除，要尽可能的balance，也就是和原来的字符串要尽可能的相似，你要怎么做（这里我们认为添加是+1 删除是-1 然后想办法让操作完之后接近0），要返回回文字符串 我一开始说dp，面试官说，别别别，直接暴力，我们暴力来看。然后我给他说我可以用DFS做，他说可以，不过我们简单一点，先考虑删除怎么做。然后我就写了删除怎么做。之后他说可以，那我们再考虑添加怎么做。我就又写了个添加怎么做。最后就是合并两个，但是时间不太够了，就大致说了一下。分析了复杂度（时间和空间）

- 做题只做了一个，比如abcdabefexy分割成abcdab efe x y，和其他的子字符串没有重复字母，思路就是**存每个字母最右边的index，然后遍历数组，不断扩大右边界，如果发现合法字符串，就截断。**
- 字符串a到b转换，每次要转换所有字母到另一个，允许用一个special char做转换桥梁，问最少转换次数from a to b

  str1 变到 str2，求最少的变次数。 规则1： a -> x的话，所有的a都需要变成x （x表示任意其他字母） 规则2：a->*，*->x

  "abc" -> "bbc" // 1, a->b "aba" -> "bbc" // -1, not possible "aba" -> "bab" // 3, a->*, b->a, *->b

  给定两个string str1,str2，判断能否将str1转换成str2，每个字符只能转换一次，每次转换所有相同字符，如 "abc"->"def" return true, "a"->"d , "b"->"e", "c"->"f" "abb"->"bcc" return true， "b"->"c", "a"->"b" "abb"->"baa" false, 无法转换-baidu 1point3acres

  follow up 1: 给定一个符号""*, 字符可以先转换成*""再转换成其他字符， "abb"->"baa" return true, "a"->""*, "b"-"a", *""->"b".

  follow up 2: 没有"*", 不限字符转换次数，即可以先将一个字符转换成中间字符，再对中间字符转换。 "abb"->"baa" return true, "a"->"c", "b"->"a", "c"->"b"

  ab->ba

  a->* *b

  b->a *a

  *->b ba

  数映射，然后找环
- 给一个迷宫, 有门有钥匙, a钥匙开A锁, 求收集齐所有钥匙后从起点到终点的最少步数。给了一个迷宫，要求求出从给定的起点走到终点的最短路径。迷宫中会有wall，key和locker。key是26个小写字母，locker是26个大写字母，遇到key要收集，遇到locker要用收集到的对应key打开，如果没有收集到对应的key，就无法通过locker。走到终点的时候必须收集到迷宫里面的所有key。楼主用bfs做的，讨论了很久如何保存中间状态，发现用bit来保存key的状态比较好。面试官各种提示，最后还没写完，所以觉得应该跪了。没想到还是发了offer。**BFS + bit存key的状态**
- 给定一个矩阵，找到两个元素，要求一个在左上方，一个在右下方，使得两个元素差值最大。
- 给了一个字符串，包含了若干大写字母和小写字母，然后要求将字符串的小写字母移到字符串的前面，大写字母移到字符串的后面，顺序无所谓。用了两个指针来做，第一个指针遍历整个字符串，第二个指针记录index最小的大写字母，每次遍历到小写字母的时候，如果index大于指针记录的大写字母的位置就交换，然后再去找下一个大写字母。**two pointers**
- 给了一列数字，要求返回每个数字的左边的最后一个比他大的数字的坐标，如果没有就返回-1。例如5 4 2 1 3，就返回-1 0 1 2 1。有点类似leetcode的739题，主要是用stack来解决。一开始是用的顺序遍历来做的，然后面试官给了一个反例，想了一会就在面试官要提示的时候想出来应该逆序来操作，然后改了下代码写出来。了。想了一下，stack那道题顺序遍历也是可以的，当遍历的数字比stack顶端小的时候就压进stack，大于等于栈顶的时候就pop掉顶部元素。可能当时有点紧张，思路不太对。**单调栈，if top < [i], keep pop(), ans = top, push**
- 有向图找圈，必须要用topological sort **算每个点的入度，每次把入度为0的放进queue**
- 给定n个圆柱体，半径和高，选K个使累加表面积最大**对半径排序，从大到小，然后dp**

- 给一个正整数集合，求一个和最大且能被3整除的子集。Follow up: 如果集合里有正有负，怎么做。用DFS是肯定可以做的，但我当时想的是先排个序，然后greedy地取集合里的所有数，看看除3余几 1) 如果余0直接return 2) 如果余1，考虑是丢掉一个最小的除3余1的数，还是丢掉两个最小的除3余2的数 3) 如果余2也是类似的 后来跟面试官讨论发现其实不用排序。。打个擂台就能找了（但其实我是想着排序了代码好写一点orz） 优化后时间复杂度是O(n)，本来还担心这个方法会不会有点野鸡，但是讲道理效率确实比DFS好得多。。。 follow up: 加入负数的话也是类似的，一开始greedy地取所有正数，然后再考虑是丢掉最小的正数还是加入最大的负数，复杂度一样
- sliding window minimum，给一个指定size的sliding window， 返回每一次的最小值 面完看了一下网上有maximum的题，也不是很难，可以做到O（n）时间 **维护一个set or 维护一个queue，最小的在前面，要是当前元素比队尾小，pop队尾**
- 给一个最小堆，每一个最小堆的节点都唯一对应一个图节点，让设计一个数据结构，使得可以修改最小堆节点的值而不改变对应关系
- 有一辆小车沿x轴运动，有两种情况会改变其速度，第一种是到达某个时间点，第二种是到达某个地点。求每个时间点所对应的x轴位移。递归解决。
- calculate possibilities (dfs -> recursion -> dp)给你n个硬币，每个硬币正面向上概率为一个array [p1, p2, ..., pn],问题：每个硬币扔一次，计算k个正面向上的概率
- 给一棵二叉树，然后每个节点多个属性叫sibling，指向同层右边第一个节点，如果右边没节点就是null。给出二叉树，让补全每个节点的sibling属性。思路是用BFS（其实就是层次遍历），然后每个节点去连队列里同一层的下一个节点。Follow up是O(1)空间，看下leetcode题解好像是有一个指针在上一层，有一个指针在下一层，然后一边遍历一边把下层的节点的sibling直接连起来，这样虽然不维护一个实际的队列，但是在访问某个节点的时候，下一个节点访问的是他的sibling，其实和层次遍历的拓展顺序是一样的。
- 给定一个BST。implementBST节点的get_weight方法，其中weight定义为此二叉树中value大于此节点value 的其他所有节点的最大深度

  input:

  5

  / \

  3 7

  / \

  1 4

  返回

  2

  / \

  3 -1(因为7为最大）

  /\

  3 2

  given a BST, weight(node i) = max depth of node j, where i.val < j.val

  val: 3 / \ 2 5. 1point3acres / / 1 4 weight: 3 / \ 3 -1 / / 3 2 **先遍历，存vector<pair<int, int>>，pair为{val， dep}对，然后对first排序，反向遍历求每个数字的最深元素。然后用map做val2dep的映射，再建树**

- longest common subsequence +follow up dp +返回一个结果 +返回所有结果
- 两堆石子(m,n)，两个人A和B，每次只能取(0,k)或者(k,0)或者(k,k)，其中k<=min(m,n)。求问如果A先取，A有没有必胜策略。首先想的是迭代，边界条件是如果m=n，那么A必定赢了。初步的想法是，f(m,n) = f(m-k,n-k) || f(m-k,n) || f(m,n-k) 等等 0<k<min(m,n)。时间复杂度是3的min(m,n)次方。后来又说能不能提升复杂度，想到了记忆化搜索。但是最后面试官说可以达到O(m)的复杂度。最后一题忘记说赢的条件了，就是谁先把两堆石子都取完了，谁就赢了
- 物体从x = 0处开始运动，初速度v0 = 1m/s, 给出两个array

  l1 = [[2,4],[4,7]...]

  l2 = [[1,3],[5,8]...]

  l1表示物体运动到l1[i][0]时刻时，速度变为l1[i][1],

  l2表示物体运动到l2[i][0]坐标时，速度变为l2[i][1].

  给定一个时间点t，求物体在t时刻的position.
- 给一个数组 裡面只有 0和1,问最少次数把0换成1 或把1换成0 可以让 0都在左边 1都在右边 (或者0都在右边 1都在左边). [0,1,0,1]的话就是把第一个1换成0 可以达到分边 **左右扫一遍，存下每个点左右各有多少个0 or 1，找到和最小的点**
- 字典树 **会写**
- 有n个object 两两之间可以有三种关系(相同种类1,不同种类2,不知道2) 给m个三元组（object1，object2，relation 1/2/3）假设不存在矛盾 求问给定一些二元组（object1，object2） 输出他们的关系

  A B 1

  A C 1

  A D 2

  C F 1

  D E 2

  则输出 B D =》 2 A E =》 3 A F =》1 并查集建点（same relation） + 图遍历建边（different relation） **union find**
- n个点在2D平面上，return k等分点的位置并返回。基本就是presum，然后考虑各种corner case，You are given a path consisting of integer points, and you want to cut it into equal k segments. Find endpoints of each segment.第二题写码，给N个点，连起来，长度K等分，找到等分点（在折线上），也不难，就是码有点难写，cornner case，dummy node 比较繁琐。medium 难度，做法就是presum 然后binary search （或者map到index也可以，我用了upperbound）到位置找点，我没仔细写，写了几行core code，也是O（N）。给n的点形成一条折线，求k等分点： 二分 **presum + 二分**
- 有n枚硬币，每个硬币掷一次正面朝上的概率各不相同，假定第i个概率为pi。如果把所有硬币都掷一次，求k个硬币正面朝上的概率。一开始想的是DFS，面试小哥说复杂度太高了，后来想到用DP。然后被问复杂度，follow up是问空间复杂度怎么优化到O(n)。**N个硬币中K个朝上可以通过N-1个硬币中K个朝上，以及N-1个硬币中K-1个朝上两个状态计算出来**
- Find a polynomial time algorithm to determine whether two trees are isomorphic
- 口头design 一个类似于找一对朋友的东西，要求朋友间距离小于K，并且朋友属于同一组，不难，用一个sliding window size K， running time O（N）。

- 平面上N个点，找出所有的正方形 **找到三个成直角的点，然后判断第四个点在不在set中**
- finding the next greater element **反向遍历数字，用一个单调栈，由底至顶递减，当前元素大于顶端元素就pop，返回元素，然后push当前元素**

  Given a circular array (the next element of the last element is the first element of the array), print the Next Greater Number for every element. The Next Greater Number of a number x is the first greater number to its traversing-order next in the array, which means you could search circularly to find its next greater number. If it doesn't exist, output -1 for this number.

```
public class Solution {

    public int[] nextGreaterElements(int[] nums) {
        int[] res = new int[nums.length];
        Stack<Integer> stack = new Stack<>();
        for (int i = 2 * nums.length - 1; i >= 0; --i) {
            while (!stack.empty() && nums[stack.peek()] <= nums[i %
nums.length]) {
                stack.pop();
            }
            res[i % nums.length] = stack.empty() ? -1 :
nums[stack.peek()];
            stack.push(i % nums.length);
        }
        return res;
    }
}

class Solution {
public:
    vector<int> nextGreaterElements(vector<int>& nums) {
        if(nums.size() == 0) return vector<int>();
        vector<int> result(nums.size(), INT_MIN);
        stack<int> stk;
        stk.push(0);
        for(int i = 1; i < nums.size(); i++){
            while(stk.empty() == false && nums[i] > nums[stk.top()]){
                result[stk.top()] = nums[i];
                stk.pop();
            }
            stk.push(i);
        }
        int i = 0;
        while(stk.size() > 1){
            while(i < nums.size() && nums[stk.top()] >= nums[i]) i++;
            if(i == nums.size()) break;
            while(nums[stk.top()] < nums[i]){
                result[stk.top()] = nums[i];
                stk.pop();
            }
```

```
        }
        while(stk.empty() == false){
            result[stk.top()] = -1;
            stk.pop();
        }
        return result;
    }
};
```

- 一个数轴上，有n个点(x1,x2,...,xn)和m个区间(a1_b1),(a2_b2),...,(am,bm)。每一个点只能匹配一个区间，每一个区间也只能匹配一个点。匹配的必要条件是点包含在区间之内. 也就是对于(ai,bi)，当ai<=x<=bi时，可以进行匹配，当然也可以选择不匹配，把这个区间让给其他的点。求最多可以有多少个区间被匹配到。可以抽象成二分图最大匹配，我觉得方法是可行的。如果面试官给我一丁点关于往图上面思考的提示就好了，我应该就能想到。可惜他没有。给n个点和m个区间，一个点最多匹配几个区间 问最多匹配几个区间和点： 堆+扫描线 需要证明正确性 貌似用二分图? **二分图匹配 -> 网络流**

```
// A DFS based recursive function
// that returns true if a matching
// for vertex u is possible
bool bpm(bool bpGraph[M][N], int u,
         bool seen[], int matchR[])
{
    // Try every job one by one
    for (int v = 0; v < N; v++)
    {
        // If applicant u is interested in
        // job v and v is not visited
        if (bpGraph[u][v] && !seen[v])
        {
            // Mark v as visited
            seen[v] = true;

            // If job 'v' is not assigned to an
            // applicant OR previously assigned
            // applicant for job v (which is matchR[v])
            // has an alternate job available.
            // Since v is marked as visited in
            // the above line, matchR[v] in the following
            // recursive call will not get job 'v' again
            if (matchR[v] < 0 || bpm(bpGraph, matchR[v],
                                     seen, matchR))
            {
                matchR[v] = u;
                return true;
            }
        }
```

```
    }
    return false;
}

// Returns maximum number
// of matching from M to N
int maxBPM(bool bpGraph[M][N])
{
    // An array to keep track of the
    // applicants assigned to jobs.
    // The value of matchR[i] is the
    // applicant number assigned to job i,
    // the value -1 indicates nobody is
    // assigned.
    int matchR[N];

    // Initially all jobs are available
    memset(matchR, -1, sizeof(matchR));

    // Count of jobs assigned to applicants
    int result = 0;
    for (int u = 0; u < M; u++)
    {
        // Mark all jobs as not seen
        // for next applicant.
        bool seen[N];
        memset(seen, 0, sizeof(seen));

        // Find if the applicant 'u' can get a job
        if (bpm(bpGraph, u, seen, matchR))
            result++;
    }
    return result;
}

// Driver Code
int main()
{
    // Let us create a bpGraph
    // shown in the above example
    bool bpGraph[M][N] = {{0, 1, 1, 0, 0, 0},
                          {1, 0, 0, 1, 0, 0},
                          {0, 0, 1, 0, 0, 0},
                          {0, 0, 1, 1, 0, 0},
                          {0, 0, 0, 0, 0, 0},
                          {0, 0, 0, 0, 0, 1}};

    cout << "Maximum number of applicants that can get job is "
         << maxBPM(bpGraph);
```

```
    return 0;
}
```

- 给定N，请通过从小到大输出，分母不超过N的真分数序列。比如给定5，就输出1/5,1/4,1/3,5/2,....等

  先开始想了一个用PriorityQueue的算法，因为不超过N的序列中，以分母归类，以n=5为例,

  1/5 2/5 3/5 4/5

  1/4 2/4 3/4

  1/3 2/3

  1/2

  首先把每一行的第一个元素加入PriorityQueue，再poll出来，poll到的数用一个指针往后移动，这样时间是O(nlogn)。后来面试官给提示说，这其实是一个从左到右，从上到下都递增的序列，每次只需要比较右边的和下边的谁大就可以了。**先生成三角矩阵，然后双指针**

-
- multiple inheritance diamond problem
- 然后问了一个栈是不是线程安全的。。我拍脑子就说不是，结果一查继承vector是安全的，面试官也没指出来。。。
- hash map index撞一起怎么办 问了Map和HashMap的实现，并问了HashMap的地址冲突的解决方案。还问了当HashMap中的地址冲突是用开放地址法解决的时候，删除一个key时的操作
- web MVC TCP/IP AJAX Socket DOM CSS React 细节，虚拟 DOM，DOM diff 算法等等

  Continous Integration 抓取网页信息的js代码，还有同步异步问题，es6和es7 的解决方案

  AJAX fetch，async await Google s2 算法

  Css JS js 的promise，异步 实现 lodash 里的 throttle 和 debounce 两个 API 闭包 + setTimeout
- docker image和container 问VPN的原理和怎么搭,多线程的概念，问程序运行时calling track怎么打log高效，他的回答是开辟一块同时进行读写操作的内存，先往内存里写然后同时往硬盘读, 为了避免冲突问要怎么加锁，如果整个空间加锁等待时间太长怎么办，然后他跟我讲要对内存进行原子操作？？
- 简历 + 多线程知识 + 实现多线程队列 + OS（进程间通信：管道 命名管道 消息队列 共享内存）问给offer去不去
- 怎么验证一个变量是空对象：{}， 我舍友说是!!就可以；第二个问题：怎么获取数据：可以用框架可以原生js：原生js写xmlhttp？ angularjs $http，还有jquery的ajax；第三个问题：有多个url，怎么快速向这么多个url中获取数据。。。我真的不知道。。。
- C++11,14,17的特性你知道有哪些吗？我：emmmm。这个不会，没关系，你知道智能指针吗？
- 给一堆点，已知他们是从一个正态分布生成的，估计平均；
- 一个游戏装备，每一轮附魔升级有一定的成功和失败几率，需要升到最大级别的期望次数，我给了个dp的解，他说跟想让我用markov解，但是这个也行。
- 字典树

- kdtree
- 给文件列表List<File> files, 实现一个gitignore, 可以过滤掉gitignore里出现的文件，需要满足gitignore的规则。gitignore规则：如果是*，则表示同通配0个或者任意字符。比如/path/to/*file。/path/to/abcdefile满足(任意)，/path/to/file满足（0）。如果是?，则表示通配一个字符。比如/path/to/?file。/path/to/zfile, /path/to/xfile等满足。如果是[]，则表示满足括号里的任意一个字符。比如/path/to/[abcde]file，则/path/to/afile, /path/to/bfile等满足。重点是，“/”不能匹配难炸了。跟leetcode 10有点像，但是比这个难多了。写了半个小时放弃。写了写可能的思路(dp)。后来面试官告诉我用正则表达式API。