[面试题 01.01. 判定字符是否唯一](https://leetcode-cn.com/problems/is-unique-lcci/)

实现一个算法，确定一个字符串 s 的所有字符是否全都不同。

**示例 1：**

**输入:** s = "leetcode"

**输出:** false

**示例 2：**

**输入:** s = "abc"

**输出:** true

class Solution {

public:

    bool isUnique(string astr) {

        int mark = 0;

        for(auto c : astr){

            if((mark & 1 << (c - 'a')) == 0)

                mark |= 1 << (c - 'a');

            else

                return false;

        }

        return true;

    }

};

[面试题 01.02. 判定是否互为字符重排](https://leetcode-cn.com/problems/check-permutation-lcci/)

给定两个字符串 s1 和 s2，请编写一个程序，确定其中一个字符串的字符重新排列后，能否变成另一个字符串。

**示例 1：**

**输入:** s1 = "abc", s2 = "bca"

**输出:** true

**示例 2：**

**输入:** s1 = "abc", s2 = "bad"

**输出:** false

class Solution {

public:

    bool CheckPermutation(string s1, string s2) {

        unordered\_map<char, int> mp;

        for(auto c : s1){

            mp[c]++;

        }

        for(auto c : s2){

            mp[c]--;

        }

        for(int i = 0; i < 26; ++i){

            char c = i + 'a';

            if(mp[c] != 0)

                return false;

        }

        return true;

    }

};

[面试题 01.03. URL化](https://leetcode-cn.com/problems/string-to-url-lcci/)

URL化。编写一种方法，将字符串中的空格全部替换为%20。假定该字符串尾部有足够的空间存放新增字符，并且知道字符串的“真实”长度。（注：用Java实现的话，请使用字符数组实现，以便直接在数组上操作。）

**示例1:**

**输入**："Mr John Smith ", 13

**输出**："Mr%20John%20Smith"

**示例2:**

**输入**：" ", 5

**输出**："%20%20%20%20%20"

class Solution {

public:

    string replaceSpaces(string S, int length) {

        string res;

        for(int i = 0; i < length; ++i){

            if(S[i] == ' ')

                res += "%20";

            else

                res += S[i];

        }

        return res;

    }

};

[面试题 01.04. 回文排列](https://leetcode-cn.com/problems/palindrome-permutation-lcci/)

给定一个字符串，编写一个函数判定其是否为某个回文串的排列之一。

回文串是指正反两个方向都一样的单词或短语。排列是指字母的重新排列。

回文串不一定是字典当中的单词。

**示例1：输入："**tactcoa"

**输出：**true（排列有"tacocat"、"atcocta"，等等）

class Solution {

public:

    bool canPermutePalindrome(string s) {

        unordered\_map<int, int> mp;

        for(auto c : s)

            mp[c]++;

        int count = 0;

        for(auto it = mp.begin(); it != mp.end(); it++){

            if(it->second % 2 == 1)

                count++;

        }

        return count < 2;

    }

};

[面试题 01.05. 一次编辑](https://leetcode-cn.com/problems/one-away-lcci/)

字符串有三种编辑操作:插入一个字符、删除一个字符或者替换一个字符。 给定两个字符串，编写一个函数判定它们是否只需要一次(或者零次)编辑。

**示例 1:**

**输入:**

first = "pale"

second = "ple"

**输出:** True

**示例 2:**

**输入:**

first = "pales"

second = "pal"

**输出:** False

class Solution {

public:

    bool oneEditAway(string first, string second) {

        if(first == second)

            return true;

        int len1 = first.size();

        int len2 = second.size();

        if(abs(len1 - len2) > 1)

            return false;

        int i = 0, j = len1 - 1, k = len2 - 1;

        while(i < len1 && i < len2 && first[i] == second[i])

            i++;

        while(j >= 0 && k >= 0 && first[j] == second[k]){

            j--;

            k--;

        }

        return j - i < 1 && k - i < 1;

        //1，插入或删除：其中短的字符串的下标相减为-1，长的字符串下标相减为0

        //2，替换，两字符串的下标相减均为0

    }

};

[面试题 01.06. 字符串压缩](https://leetcode-cn.com/problems/compress-string-lcci/)

字符串压缩。利用字符重复出现的次数，编写一种方法，实现基本的字符串压缩功能。比如，字符串aabcccccaaa会变为a2b1c5a3。若“压缩”后的字符串没有变短，则返回原先的字符串。你可以假设字符串中只包含大小写英文字母（a至z）。

**示例1:**

**输入**："aabcccccaaa"

**输出**："a2b1c5a3"

**示例2:**

**输入**："abbccd"

**输出**："abbccd"

**解释**："abbccd"压缩后为"a1b2c2d1"，比原字符串长度更长。

class Solution {

public:

    string compressString(string S) {

        if(!S.size())

            return S;

        string ans = "";

        int cnt = 1;

        char c = S[0];

        for(int i = 1; i < S.size(); ++i){

            if(c == S[i]){

                ++cnt;

            }

            else{

                ans += c + to\_string(cnt);

                c = S[i];

                cnt = 1;

            }

        }

        ans += c + to\_string(cnt);

        return ans.size() < S.size() ? ans : S;

    }

};

[面试题 01.07. 旋转矩阵](https://leetcode-cn.com/problems/rotate-matrix-lcci/)

给你一幅由 N × N 矩阵表示的图像，其中每个像素的大小为 4 字节。请你设计一种算法，将图像旋转 90 度。

不占用额外内存空间能否做到？

**示例 1:**

给定 **matrix** =

[

[1,2,3],

[4,5,6],

[7,8,9]

],

**原地**旋转输入矩阵，使其变为:

[

[7,4,1],

[8,5,2],

[9,6,3]

]

**示例 2:**

给定 **matrix** =

[

[ 5, 1, 9,11],

[ 2, 4, 8,10],

[13, 3, 6, 7],

[15,14,12,16]

],

**原地**旋转输入矩阵，使其变为:

[

[15,13, 2, 5],

[14, 3, 4, 1],

[12, 6, 8, 9],

[16, 7,10,11]

]

//一、使用辅助数组

class Solution {

public:

    void rotate(vector<vector<int>>& matrix) {

        int n = matrix.size();

        // C++ 这里的 = 拷贝是值拷贝，会得到一个新的数组

        auto matrix\_new = matrix;

        for (int i = 0; i < n; ++i) {

            for (int j = 0; j < n; ++j) {

                matrix\_new[j][n - i - 1] = matrix[i][j];

            }

        }

        // 这里也是值拷贝

        matrix = matrix\_new;

    }

};

// 复杂度分析

// 时间复杂度：O(N^2)，其中 N 是 matrix 的边长。

// 空间复杂度：O(N^2)，我们需要使用一个和 matrix 的大小相同的辅助数组。

//二、原地旋转，画一下图(0, 1) (1, 3)....就知道了

class Solution {

public:

    void rotate(vector<vector<int>>& matrix) {

        int n = matrix.size();

        for(int i = 0; i < n / 2; ++i)

            for(int j = i; j < n - 1 - i; ++j){

                int temp = matrix[i][j];

                matrix[i][j] = matrix[n - 1 - j][i];

                matrix[n - 1 - j][i] = matrix[n - 1 -i][n - 1 - j];

                matrix[n - 1 -i][n - 1 -j] = matrix[j][n - 1 - i];

                matrix[j][n - 1 - i] = temp;

            }

    }

};

//O(N^2)  O(1)

//方法三：用翻转代替旋转

class Solution {

public:

    void rotate(vector<vector<int>>& matrix) {

        int n = matrix.size();

        // 水平翻转

        for (int i = 0; i < n / 2; ++i) {

            for (int j = 0; j < n; ++j) {

                swap(matrix[i][j], matrix[n - i - 1][j]);

            }

        }

        // 主对角线翻转

        for (int i = 0; i < n; ++i) {

            for (int j = 0; j < i; ++j) {

                swap(matrix[i][j], matrix[j][i]);

            }

        }

    }

};

[面试题 01.09. 字符串轮转](https://leetcode-cn.com/problems/string-rotation-lcci/)

字符串轮转。给定两个字符串s1和s2，请编写代码检查s2是否为s1旋转而成（比如，waterbottle是erbottlewat旋转后的字符串）。

**示例1:**

**输入**：s1 = "waterbottle", s2 = "erbottlewat"

**输出**：True

**示例2:**

**输入**：s1 = "aa", s2 = "aba"

**输出**：False

class Solution {

public:

    bool isFlipedString(string s1, string s2) {

        return s1.size() == s2.size() && (s1 + s1).find(s2) != -1;

    }

};

//find函数的时间复杂度还是O(MN);

[面试题 02.01. 移除重复节点](https://leetcode-cn.com/problems/remove-duplicate-node-lcci/)

编写代码，移除未排序链表中的重复节点。保留最开始出现的节点。

**示例1:**

**输入**：[1, 2, 3, 3, 2, 1]

**输出**：[1, 2, 3]

**示例2:**

**输入**：[1, 1, 1, 1, 2]

**输出**：[1, 2]

struct ListNode{

    int val;

    ListNode\* next;

    ListNode(int x) : val(x), next(NULL) {}

};

class Solution {

public:

    ListNode\* removeDuplicateNodes(ListNode\* head) {

        if (head == nullptr) {

            return head;

        }

        unordered\_set<int> occurred = {head->val};

        ListNode\* pos = head;

        // 枚举前驱节点

        while (pos->next != nullptr) {

            // 当前待删除节点

            ListNode\* cur = pos->next;

            if (!occurred.count(cur->val)) {

                occurred.insert(cur->val);

                pos = pos->next;

            } else {

                pos->next = pos->next->next;

            }

        }

        pos->next = nullptr;

        return head;

    }

};

// 复杂度分析

// 时间复杂度：O(N)，其中 N是给定链表中节点的数目。

// 空间复杂度：O(N)。在最坏情况下，给定链表中每个节点都不相同，哈希表中需要存储所有的 N个值。

class Solution {

public:

    ListNode\* removeDuplicateNodes(ListNode\* head) {

        ListNode\* ob = head;

        while (ob != nullptr) {

            ListNode\* oc = ob;

            while (oc->next != nullptr) {

                if (oc->next->val == ob->val) {

                    oc->next = oc->next->next;

                } else {

                    oc = oc->next;

                }

            }

            ob = ob->next;

        }

        return head;

    }

};

// 复杂度分析

// 时间复杂度：O(N^2)，其中 N是给定链表中节点的数目。空间复杂度：O(1)。

[面试题 02.02. 返回倒数第 k 个节点](https://leetcode-cn.com/problems/kth-node-from-end-of-list-lcci/)

实现一种算法，找出单向链表中倒数第 k 个节点。返回该节点的值。

**注意：**本题相对原题稍作改动

**示例：**

**输入：** 1->2->3->4->5 和 *k* = 2

**输出：** 4

class Solution {

public:

    int kthToLast(ListNode\* head, int k) {

        ListNode \*p = head, \*q = head; //初始化

        while(k--) {   //将 p指针移动 k 次

            p = p->next;

        }

        while(p != nullptr) {//同时移动，直到 p == nullptr

            p = p->next;

            q = q->next;

        }

        return q->val;

    }

};

[面试题 02.03. 删除中间节点](https://leetcode-cn.com/problems/delete-middle-node-lcci/)

实现一种算法，删除单向链表中间的某个节点（即不是第一个或最后一个节点），假定你只能访问该节点。

**示例：输入：**单向链表a->b->c->d->e->f中的节点c

**结果：**不返回任何数据，但该链表变为a->b->d->e->f

class Solution {

public:

    void deleteNode(ListNode\* node) {

        //将容貌换成后面节点的样子（1.狸猫易容）

        node->val=node->next->val;

        //借助指针p将node->next指向的那块堆空间释放（为了保证代码的健壮性）

        ListNode \*p=node->next;

        //取而代之(2.狸猫换太子)

        node->next=node->next->next;

        //防止内存泄漏（为了保证代码的健壮性）

        delete p;

    }

};

[面试题 02.04. 分割链表](https://leetcode-cn.com/problems/partition-list-lcci/)

编写程序以 x 为基准分割链表，使得所有小于 x 的节点排在大于或等于 x 的节点之前。如果链表中包含 x，x 只需出现在小于 x 的元素之后(如下所示)。分割元素 x 只需处于“右半部分”即可，其不需要被置于左右两部分之间。

**示例:**

**输入:** head = 3->5->8->5->10->2->1, *x* = 5

**输出:** 3->1->2->10->5->5->8

class Solution {

public:

    ListNode\* partition(ListNode\* head, int x) {

        ListNode\* head1 = new ListNode(0);

        ListNode\* head2 = new ListNode(0);

        ListNode\* p = head,\* p1 = head1,\* p2 = head2;

        while(p){

            if(p->val < x){

                p1->next = p;

                p1 = p1->next;

            }else{

                p2->next = p;

                p2 = p2->next;

            }

            p = p->next;

        }

        p1->next = head2->next;

        p2->next = nullptr;

        return head1->next;

    }

};//O(n)//O(1)

[面试题 02.05. 链表求和](https://leetcode-cn.com/problems/sum-lists-lcci/)

给定两个用链表表示的整数，每个节点包含一个数位。

这些数位是反向存放的，也就是个位排在链表首部。

编写函数对这两个整数求和，并用链表形式返回结果。

**示例：**

**输入：**(7 -> 1 -> 6) + (5 -> 9 -> 2)，即617 + 295

**输出：**2 -> 1 -> 9，即912

**进阶：**假设这些数位是正向存放的，请再做一遍。

**示例：**

**输入：**(6 -> 1 -> 7) + (2 -> 9 -> 5)，即617 + 295

**输出：**9 -> 1 -> 2，即912

struct ListNode {

     int val;

     ListNode \*next;

     ListNode(int x) : val(x), next(NULL) {}

};

class Solution

{

public:

    ListNode\* addTwoNumbers(ListNode\* l1, ListNode\* l2)

    {

        ListNode \*head = new ListNode(-1), \*p1 = l1, \*p2 = l2, \*p = head;//用带头节点的可以少一点初始的特判

        int sum = 0, carr = 0;

        while (p1 || p2 || carr) //如果改用&&则while结束还要多一些特判

        {

            sum = 0;//当前两位数字和

            if(p1)

            {

                sum += (p1->val);

                p1 = p1->next;

            }

            if(p2)

            {

                sum += (p2->val);

                p2 = p2->next;

            }

            sum += carr; //加上上一位的进位

            ListNode \*t = new ListNode(sum % 10); //得到当前位数字

            carr = sum / 10; //得到当前位对下一位的进位

            p->next = t;//当前位连接上去

            p = p->next;//游标指针更新

        }

        return head->next;

    }

};

[面试题 02.06. 回文链表](https://leetcode-cn.com/problems/palindrome-linked-list-lcci/)

编写一个函数，检查输入的链表是否是回文的。

**示例 1：**

**输入：** 1->2

**输出：** false

**示例 2：**

**输入：** 1->2->2->1

**输出：** true

class Solution {

public:

//通过快慢指针找到中间结点

//通过头插法将链表前半段反转

//最后比较

    bool isPalindrome(ListNode\* head) {

        if(head == NULL)

            return true;

        ListNode\* fast = head;

        ListNode\* slow = head;

        while(fast && fast->next){//快慢指针找中间结点

            fast = fast->next->next;

            slow = slow->next;

        }

        ListNode\* p = head;

        ListNode\* dummyHead = new ListNode(-1);

        dummyHead->next = head;

        while(p->next != slow && p->next != NULL){//头插法反转前半段

            ListNode\* r = p->next;

            p->next = r->next;

            r->next = dummyHead->next;

            dummyHead->next = r;

        }

        p = dummyHead->next;

        if(fast)//如果慢指针指向正中间的结点

            slow = slow->next;

        while(slow){//比较是否为回文

            if(slow->val != p->val)

                return false;

            slow = slow->next;

            p = p->next;

        }

        return true;

    }

};

面试题 02.07. 链表相交

给定两个（单向）链表，判定它们是否相交并返回交点。请注意相交的定义基于节点的引用，而不是基于节点的值。换句话说，如果一个链表的第k个节点与另一个链表的第j个节点是同一节点（引用完全相同），则这两个链表相交。

class Solution {

public:

    ListNode \*getIntersectionNode(ListNode \*headA, ListNode \*headB) {

        ListNode \*node1 = headA;

        ListNode \*node2 = headB;

        while(node1 != node2){

            node1 = node1 != NULL ? node1->next : headB;

            node2 = node2 != NULL ? node2->next : headA;

        }

        return node1;

    }

};

[面试题 02.08. 环路检测](https://leetcode-cn.com/problems/linked-list-cycle-lcci/)

给定一个链表，如果它是有环链表，实现一个算法返回环路的开头节点。  
有环链表的定义：在链表中某个节点的next元素指向在它前面出现过的节点，则表明该链表存在环路。

**示例 1：**

**输入：**head = [3,2,0,-4], pos = 1

**输出：**tail connects to node index 1

**解释：**链表中有一个环，其尾部连接到第二个节点。

**示例 2：**

**输入：**head = [1], pos = -1

**输出：**no cycle

**解释：**链表中没有环。

class Solution {

public:

    ListNode \*detectCycle(ListNode \*head) {

        if(!head||!head->next)

        {

            return NULL;

        }

        ListNode\* fast=head;

        ListNode\* slow=head;

        while(fast&&fast->next)

        {

            slow=slow->next;

            fast=fast->next->next;

            if(fast==slow)

            {

                break;

            }

        }

        if(fast!=slow)

        {

            return NULL;

        }

        fast=head;

        while(fast!=slow)

        {

            fast=fast->next;

            slow=slow->next;

        }

        return fast;

    }

};

[面试题 03.01. 三合一](https://leetcode-cn.com/problems/three-in-one-lcci/)

三合一。描述如何只用一个数组来实现三个栈。

你应该实现push(stackNum, value)、pop(stackNum)、isEmpty(stackNum)、peek(stackNum)方法。stackNum表示栈下标，value表示压入的值。

构造函数会传入一个stackSize参数，代表每个栈的大小。

**示例1:输入**：

["TripleInOne", "push", "push", "pop", "pop", "pop", "isEmpty"]

[[1], [0, 1], [0, 2], [0], [0], [0], [0]]

**输出**：[null, null, null, 1, -1, -1, true]

**说明**：当栈为空时`pop, peek`返回-1，当栈满时`push`不压入元素。

class TripleInOne {

public:

    int \*stack;

    int top[3];

    int stackSize;

    TripleInOne(int stackSize):stackSize(stackSize) {

        stack = new int[stackSize\*3];

        top[0]=top[1]=top[2]=0;

    }

    void push(int stackNum, int value) {

        if(top[stackNum] < stackSize)

            stack[stackSize\*stackNum + top[stackNum]++]=value;

    }

    int pop(int stackNum) {

        if(top[stackNum] <= 0)

            return -1;

        else

            return stack[stackSize\*stackNum + (--top[stackNum])];

    }

    int peek(int stackNum) {

        if(top[stackNum] <= 0)

            return -1;

        else

            return stack[stackSize\*stackNum + (top[stackNum]-1)];

    }

    bool isEmpty(int stackNum) {

        return top[stackNum]==0;

    }

};

[面试题 03.02. 栈的最小值](https://leetcode-cn.com/problems/min-stack-lcci/)

请设计一个栈，除了常规栈支持的pop与push函数以外，还支持min函数，该函数返回栈元素中的最小值。执行push、pop和min操作的时间复杂度必须为O(1)。

**示例：**

MinStack minStack = new MinStack();  
minStack.push(-2);  
minStack.push(0);  
minStack.push(-3);  
minStack.getMin(); --> 返回 -3.  
minStack.pop();  
minStack.top(); --> 返回 0.  
minStack.getMin(); --> 返回 -2.

class MinStack {

public:

    stack<pair<int, int>> s;

    MinStack() {

    }

    void push(int x) {

        if(s.empty())

            s.push(make\_pair(x, x));

        else

            s.push(make\_pair(x, min(x, s.top().second)));

   }

   void pop() {

        s.pop();

    }

    int top() {

        return s.top().first;

    }

    int getMin() {

        return s.top().second;

    }

};

[面试题 03.03. 堆盘子](https://leetcode-cn.com/problems/stack-of-plates-lcci/)

堆盘子。设想有一堆盘子，堆太高可能会倒下来。因此，在现实生活中，盘子堆到一定高度时，我们就会另外堆一堆盘子。请实现数据结构SetOfStacks，模拟这种行为。SetOfStacks应该由多个栈组成，并且在前一个栈填满时新建一个栈。此外，SetOfStacks.push()和SetOfStacks.pop()应该与普通栈的操作方法相同（也就是说，pop()返回的值，应该跟只有一个栈时的情况一样）。 进阶：实现一个popAt(int index)方法，根据指定的子栈，执行pop操作。

当某个栈为空时，应当删除该栈。当栈中没有元素或不存在该栈时，pop，popAt 应返回 -1.

**示例1:输入**：

["StackOfPlates", "push", "push", "popAt", "pop", "pop"]

[[1], [1], [2], [1], [], []]

**输出**：[null, null, null, 2, 1, -1]

**示例2:输入**：

["StackOfPlates", "push", "push", "push", "popAt", "popAt", "popAt"]

[[2], [1], [2], [3], [0], [0], [0]]

**输出**：[null, null, null, null, 2, 1, 3]

class StackOfPlates {

 private:

    int size;

    vector<stack<int> > stack\_sets;

public:

    StackOfPlates(int cap) {

        size=cap;

    }

    void push(int val) {

        if(size==0)  return;

        if(stack\_sets.size()==0||stack\_sets[stack\_sets.size()-1].size()==size)

        {

            stack<int> tmp;

            tmp.push(val);

            stack\_sets.push\_back(tmp);

        }

        else

            stack\_sets[stack\_sets.size()-1].push(val);

    }

    int pop() {

        if(stack\_sets.size()==0) return -1;

        int ret=stack\_sets[stack\_sets.size()-1].top();

        stack\_sets[stack\_sets.size()-1].pop();

        if(stack\_sets[stack\_sets.size()-1].empty())

           {

                auto it=stack\_sets.end();

                it--;

            stack\_sets.erase(it);

        }

        return ret;

    }

    int popAt(int index) {

        if(stack\_sets.size()==0||index<0||index>=stack\_sets.size())

            return -1;

        int ret=stack\_sets[index].top();

        stack\_sets[index].pop();

        if(stack\_sets[index].empty())

            stack\_sets.erase(stack\_sets.begin()+index);

        return ret;

    }

};

[面试题 03.04. 化栈为队](https://leetcode-cn.com/problems/implement-queue-using-stacks-lcci/)

实现一个MyQueue类，该类用两个栈来实现一个队列。

**示例：**

MyQueue queue = new MyQueue();  
  
queue.push(1);  
queue.push(2);  
queue.peek(); // 返回 1  
queue.pop(); // 返回 1  
queue.empty(); // 返回 false

class MyQueue {

    stack<int> s1, s2;

public:

    MyQueue() {

    }

    void push(int x) {

        s1.push(x);

    }

    int pop() {

        int val = peek();

        s2.pop();

        return val;

    }

    int peek() {

        if(s2.empty()){

            while(!s1.empty()){

                int val = s1.top(); s1.pop();

                s2.push(val);

            }

        }

        return s2.top();

    }

    bool empty() {

        return s1.empty() && s2.empty();

    }

};

[面试题 03.05. 栈排序](https://leetcode-cn.com/problems/sort-of-stacks-lcci/)

栈排序。 编写程序，对栈进行排序使最小元素位于栈顶。最多只能使用一个其他的临时栈存放数据，但不得将元素复制到别的数据结构（如数组）中。该栈支持如下操作：push、pop、peek 和 isEmpty。当栈为空时，peek 返回 -1。

**示例1:**

**输入**：

["SortedStack", "push", "push", "peek", "pop", "peek"]

[[], [1], [2], [], [], []]

**输出**：

[null,null,null,1,null,2]

**示例2:**

**输入**：

["SortedStack", "pop", "pop", "push", "pop", "isEmpty"]

[[], [], [], [1], [], []]

**输出**：

[null,null,null,null,null,true]

class SortedStack {

public:

    stack<int> s1, s2;

    SortedStack() {

    }

    void push(int val) {

        while(!s1.empty() && s1.top() < val){

            s2.push(s1.top());

            s1.pop();

        }

        s1.push(val);

        while(!s2.empty()){

            s1.push(s2.top());

            s2.pop();

        }

    }

    void pop() {

        if(!s1.empty())

            s1.pop();

    }

    int peek() {

        if(!s1.empty())

            return s1.top();

        return -1;

    }

    bool isEmpty() {

        return s1.empty();

    }

};

class SortedStack {

public:

    stack<int>s1;//原栈为降序

    stack<int>s2;//辅助栈为升序

    SortedStack() {

    }

    void push(int val) {

        while(!s2.empty() && s2.top() > val){//辅助栈中存在比val大的值

            s1.push(s2.top());

            s2.pop();

        }

        while(!s1.empty() && s1.top() < val){//原栈中有比val小的值

            s2.push(s1.top());

            s1.pop();

        }

        s1.push(val);

    }

    void pop() {

        while(!s2.empty()){//清空辅助栈

            s1.push(s2.top());

            s2.pop();

        }

        if(!s1.empty()) s1.pop();

    }

    int peek() {

        while(!s2.empty()){//清空辅助栈

            s1.push(s2.top());

            s2.pop();

        }

        if(!s1.empty()) return s1.top();

        else return -1;

    }

    bool isEmpty() {

        return s1.empty() && s2.empty();

    }

};

[面试题 03.06. 动物收容所](https://leetcode-cn.com/problems/animal-shelter-lcci/)

动物收容所。有家动物收容所只收容狗与猫，且严格遵守“先进先出”的原则。在收养该收容所的动物时，收养人只能收养所有动物中“最老”（由其进入收容所的时间长短而定）的动物，或者可以挑选猫或狗（同时必须收养此类动物中“最老”的）。换言之，收养人不能自由挑选想收养的对象。请创建适用于这个系统的数据结构，实现各种操作方法，比如enqueue、dequeueAny、dequeueDog和dequeueCat。允许使用Java内置的LinkedList数据结构。

enqueue方法有一个animal参数，animal[0]代表动物编号，animal[1]代表动物种类，其中 0 代表猫，1 代表狗。

dequeue\*方法返回一个列表[动物编号, 动物种类]，若没有可以收养的动物，则返回[-1,-1]。

**示例1:**

**输入**：

["AnimalShelf", "enqueue", "enqueue", "dequeueCat", "dequeueDog", "dequeueAny"]

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

**输出**：

[null,null,null,[0,0],[-1,-1],[1,0]]

**示例2:**

**输入**：

["AnimalShelf", "enqueue", "enqueue", "enqueue", "dequeueDog", "dequeueCat", "dequeueAny"]

[[], [[0, 0]], [[1, 0]], [[2, 1]], [], [], []]

**输出**：

[null,null,null,null,[2,1],[0,0],[1,0]]

class AnimalShelf {

public:

    AnimalShelf() {

    }

    void enqueue(vector<int> animal) {

        if (animal.at(1) == 0) //动物种类：猫

            m\_queue\_cat.push(animal);

        else

            m\_queue\_dog.push(animal);

    }

    vector<int> dequeueAny() {

        vector<int> animal;

        if (m\_queue\_cat.empty() && m\_queue\_dog.empty()){

            return vector<int>{-1, -1};

        }

        else if (!m\_queue\_cat.empty() && m\_queue\_dog.empty()){

            animal = m\_queue\_cat.front();

            m\_queue\_cat.pop();

        }

        else if (m\_queue\_cat.empty() && !m\_queue\_dog.empty()){

            animal = m\_queue\_dog.front();

            m\_queue\_dog.pop();

        }

        else{

            if (m\_queue\_cat.front().at(0) < m\_queue\_dog.front().at(0)){ //动物编号：猫 < 狗 猫先出

                animal = m\_queue\_cat.front();

                m\_queue\_cat.pop();

            }

            else{

                animal = m\_queue\_dog.front();

                m\_queue\_dog.pop();

            }

        }

        return animal;

    }

    vector<int> dequeueDog() {

        if (m\_queue\_dog.empty())

            return vector<int>{-1, -1};

        vector<int> dog = m\_queue\_dog.front();

        m\_queue\_dog.pop();

        return dog;

    }

    vector<int> dequeueCat() {

        if (m\_queue\_cat.empty())

            return vector<int>{-1, -1};

        vector<int> cat = m\_queue\_cat.front();

        m\_queue\_cat.pop();

        return cat;

    }

private:

    queue<vector<int>> m\_queue\_cat;

    queue<vector<int>> m\_queue\_dog;

};

[面试题 04.01. 节点间通路](https://leetcode-cn.com/problems/route-between-nodes-lcci/)

节点间通路。给定有向图，设计一个算法，找出两个节点之间是否存在一条路径。

**示例1:输入**：n = 3, graph = [[0, 1], [0, 2], [1, 2], [1, 2]], start = 0, target = 2

**输出**：true

**示例2:输入**：n = 5, graph = [[0, 1], [0, 2], [0, 4], [0, 4], [0, 1], [1, 3], [1, 4], [1, 3], [2, 3], [3, 4]], start = 0, target = 4

**输出** true

class Solution {

    vector<bool> viewed;

    vector<vector<int>> adList;

public:

    bool findWhetherExistsPath(int n, vector<vector<int>>& graph, int start, int target) {

        viewed = vector<bool>(n,0);

        adList = vector(n,vector<int>(1,-1));

        for (int i=0;i<graph.size();i++){

            adList[graph[i][0]].push\_back(graph[i][1]);

        }

        return search(start,target);

    }

    bool search(int start,int target){

        viewed[start] = 1;

        bool result = 0;

        for(int i=1;i<adList[start].size();i++){

            if (viewed[adList[start][i]]==0){

                if(adList[start][i]==target){

                    viewed[adList[start][i]] =1 ;

                    return 1;

                }

               result = search(adList[start][i],target);

                if(result==1)

                    break;

            }

        }

        return result;

    }

bool search(int start,int target){

    queue<int> q;

    int v;

    viewed[start] =1;

    q.push(start);

    while(!q.empty()){

        v = q.front();

        q.pop();

        for(int i=1;i<adList[v].size();i++){

            if(viewed[adList[v][i]]==0){

                viewed[adList[v][i]] = 1;

                if(adList[v][i]==target){

                    return 1;

                }

                else{

                    q.push(adList[v][i]);

                }

            }

        }

    }

    return 0;

}

};

[面试题 04.02. 最小高度树](https://leetcode-cn.com/problems/minimum-height-tree-lcci/)

给定一个有序整数数组，元素各不相同且按升序排列，编写一个算法，创建一棵高度最小的二叉搜索树。

**示例:**给定有序数组: [-10,-3,0,5,9],  
一个可能的答案是：[0,-3,9,-10,null,5]，它可以表示下面这个高度平衡二叉搜索树：  
 0   
 / \   
 -3 9   
 / /   
 -10 5

struct TreeNode {

     int val;

     TreeNode \*left;

    TreeNode \*right;

     TreeNode(int x) : val(x), left(NULL), right(NULL) {}

 };

class Solution {

public:

    TreeNode\* sortedArrayToBST(vector<int>& nums) {

        return help(nums, 0, nums.size() - 1);

    }

    TreeNode\* help(vector<int>& nums, int low, int high){

        if(low > high) // low > high表示子数组为空

            return NULL;

        // 以mid作为根节点

        int mid = (high + low) >> 1;

        TreeNode \*root = new TreeNode(nums[mid]);

        // 左子数组[low, mid -1]构建左子树

        root->left = help(nums, low, mid - 1);

        // 右子数组[mid + 1, high]构建右子树

        root->right = help(nums, mid + 1, high);

        return root;

    }

};

[面试题 04.03. 特定深度节点链表](https://leetcode-cn.com/problems/list-of-depth-lcci/)

给定一棵二叉树，设计一个算法，创建含有某一深度上所有节点的链表（比如，若一棵树的深度为 D，则会创建出 D 个链表）。返回一个包含所有深度的链表的数组。

**示例：输入：**[1,2,3,4,5,null,7,8]

1

/ \

2 3

/ \ \

4 5 7

/

8

**输出：**[[1],[2,3],[4,5,7],[8]]

class Solution {

public:

    vector<ListNode\*> listOfDepth(TreeNode\* tree) {

        queue<TreeNode\*> q;

        q.push(tree);

        vector<ListNode\*> ret;

        while (!q.empty()) {

            int sz = q.size();

            ListNode\* head = new ListNode(0);

            ListNode\* p = head;

            while (sz--) {

                TreeNode\* cur = q.front();

                q.pop();

                if (cur->left) {

                    q.push(cur->left);

                }

                if (cur->right) {

                    q.push(cur->right);

                }

                p->next = new ListNode(cur->val);

                p = p->next;

            }

            ret.push\_back(head->next);

            delete head;

        }

        return ret;

    }

};

[面试题 04.04. 检查平衡性](https://leetcode-cn.com/problems/check-balance-lcci/)

实现一个函数，检查二叉树是否平衡。在这个问题中，平衡树的定义如下：任意一个节点，其两棵子树的高度差不超过 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 。

class Solution {

public:

    bool isBalanced(TreeNode\* root) {

        return recur(root) != -1;

    }

    int recur(TreeNode\* root){

        if(root == NULL) return 0;

        int left = recur(root->left);

        if(left == -1)

            return -1;

        int right = recur(root->right);

        if(right == -1)

            return -1;

        return abs(left - right) < 2 ? max(left, right) + 1 : -1;

    }

};

class Solution {

public:

    bool isBalanced(TreeNode\* root) {

        if(root == NULL)

            return true;

        return abs(depth(root->left) - depth(root->right)) <= 1 && isBalanced(root->left) && isBalanced(root->right);

    }

    int depth(TreeNode\* root){

        if(root == NULL)

            return 0;

        return max(depth(root->left), depth(root->right)) + 1;

    }

};

[面试题 04.05. 合法二叉搜索树](https://leetcode-cn.com/problems/legal-binary-search-tree-lcci/)

实现一个函数，检查一棵二叉树是否为二叉搜索树。

**示例 1:**

**输入:**  
 2  
 / \  
 1 3  
**输出:** true

**示例 2:**

**输入:**

5  
 / \  
 1 4  
  / \  
  3 6  
**输出:** false  
**解释:** 输入为: [5,1,4,null,null,3,6]。  
  根节点的值为 5 ，但是其右子节点值为 4 。

//二叉搜索树满足中序遍历为有序序列的性质，所以直接中序遍历判断是否有序即可。

class Solution {

public:

    vector<int> ans;

    void visit(TreeNode\* root) {

        ans.push\_back(root->val);

    }

    void dfs(TreeNode\* root) {

        if (root != NULL) {

            dfs(root->left);

            visit(root);

            dfs(root->right);

        }

    }

    bool isValidBST(TreeNode\* root) {

        if (root == NULL)

            return true;

        dfs(root);

        for (int i = 0; i < ans.size() - 1; i++) {

            if (ans[i + 1] <= ans[i])

                return false;

        }

        return true;

    }

};

[面试题 04.06. 后继者](https://leetcode-cn.com/problems/successor-lcci/)

设计一个算法，找出二叉搜索树中指定节点的“下一个”节点（也即中序后继）。

如果指定节点没有对应的“下一个”节点，则返回null。

**示例 1:**

**输入:** root = [2,1,3], p = 1

2

/ \

1 3

**输出:** 2

**示例 2:**

**输入:** root = [5,3,6,2,4,null,null,1], p = 6

5

/ \

3 6

/ \

2 4

/

1

**输出:** null

BST+递归

首先本题中的二叉树还是个二叉搜索树，也就是中序遍历是单调递增的，所以我们可以利用这个性质来简化查找过程。

如果结点 p 的值大于等于 root 的值，说明 p 的后继结点在 root 右子树中，那么就递归到右子树中查找。

如果结点 p 的值小于 root 的值，说明 p 在 root 左子树中，而它的后继结点有两种可能，要么也在左子树中，要么就是 root：

如果左子树中找到了后继结点，那就直接返回答案。

如果左子树中没有找到后继结点，那就说明 p 的右儿子为空，那么 root 就是它的后继结点。

BST+非递归

如果 p 有右儿子，那么它的后继结点就是右子树的最左边的儿子。

如果 p 没有右儿子，那么它的后继结点就是，沿着 p 往上到 root 的路径中，第一个左儿子在路径上的结点。因为这个结点的左子树中 p 是最右边的结点，是最大的，所以它就是 p 的后继结点。因为是二叉搜索树，我们就可以从根结点开始往 p 走，根据结点值的大小决定走的方向。

一般树+递归

那如果是一般的二叉树，中序遍历就不满足单调递增了，这时候我们就只能找出中序遍历的结点顺序，然后才能得到 p 的后继结点。所以我们直接采用递归来做中序遍历就行了，中序遍历结果保存下来，最后取 p 的下一个结点。

一般树+非递归

当然还可以采用栈来做中序遍历，这样就是非递归了。同样结果保存下来，最后取 p 的下一个结点。

一般树+Morris遍历

 Morris 遍历算法，用常数空间来解决结点无法访问父结点的问题。这里就不细讲了，请直接看之前的题解。方法是一样的，用 Morris 遍历得到中序遍历，然后遍历一遍找到 p ，输出它的下一个结点就行了。

BST+递归（c++）

class Solution {

public:

    TreeNode\* inorderSuccessor(TreeNode\* root, TreeNode\* p) {

        if (root == NULL || p == NULL) return NULL;

        if (p->val >= root->val) {

            return inorderSuccessor(root->right, p);

        } else {

            TreeNode\* left = inorderSuccessor(root->left, p);

            return left ? left : root;

        }

    }

};

BST+非递归（c++）

class Solution {

public:

    TreeNode\* inorderSuccessor(TreeNode\* root, TreeNode\* p) {

        if (p->right) {

            p = p->right;

            while (p->left) p = p->left;

            return p;

        }

        TreeNode\* res = NULL;

        while (root != p) {

            if (root->val < p->val) {

                root = root->right;

            } else {

                res = root;

                root = root->left;

            }

        }

        return res;

    }

};

一般树+递归（c++）

class Solution {

public:

    void inorder(TreeNode\* root, vector<TreeNode\*>& res) {

        if (root->left) inorder(root->left, res);

        res.push\_back(root);

        if (root->right) inorder(root->right, res);

    }

    TreeNode\* inorderSuccessor(TreeNode\* root, TreeNode\* p) {

        vector<TreeNode\*> res;

        inorder(root, res);

        res.push\_back(NULL);

        for (int i = 0; i < res.size(); ++i) {

            if (res[i] == p) {

                return res[i+1];

            }

        }

        return NULL;

    }

};

一般树+非递归（c++）

class Solution {

public:

    TreeNode\* inorderSuccessor(TreeNode\* root, TreeNode\* p) {

        vector<TreeNode\*> res;

        stack<TreeNode\*> st;

        while (!st.empty() || root) {

            while (root) {

                st.push(root);

                root = root->left;

            }

            root = st.top();

            st.pop();

            res.push\_back(root);

            root = root->right;

        }

        res.push\_back(NULL);

        for (int i = 0; i < res.size(); ++i) {

            if (res[i] == p) {

                return res[i+1];

            }

        }

        return NULL;

    }

};

一般树+Morris遍历（c++）

class Solution {

public:

    TreeNode\* inorderSuccessor(TreeNode\* root, TreeNode\* p) {

        vector<TreeNode\*> res;

        TreeNode \*rightmost = NULL;

        while (root) {

            if (root->left) {

                rightmost = root->left;

                while (rightmost->right && rightmost->right != root) {

                    rightmost = rightmost->right;

                }

                if (rightmost->right != root) {

                    rightmost->right = root;

                    root = root->left;

                } else {

                    res.push\_back(root);

                    rightmost->right = NULL;

                    root = root->right;

                }

            } else {

                res.push\_back(root);

                root = root->right;

            }

        }

        res.push\_back(NULL);

        for (int i = 0; i < res.size(); ++i) {

            if (res[i] == p) {

                return res[i+1];

            }

        }

        return NULL;

    }

};

[面试题 04.08. 首个共同祖先](https://leetcode-cn.com/problems/first-common-ancestor-lcci/)

设计并实现一个算法，找出二叉树中某两个节点的第一个共同祖先。不得将其他的节点存储在另外的数据结构中。注意：这不一定是二叉搜索树。

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

3

/ \

5 1

/ \ / \

6 2 0 8

/ \

7 4

示例 1:

输入: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 1

输出: 3

解释: 节点 5 和节点 1 的最近公共祖先是节点 3。

class Solution {

public:

TreeNode\* lowestCommonAncestor(TreeNode\* root, TreeNode\* p, TreeNode\* q) {

        if(root == NULL || root == p || root == q)

            return root;

        TreeNode\* left = lowestCommonAncestor(root->left, p, q);

        TreeNode\* right = lowestCommonAncestor(root->right, p, q);

        if(left == NULL)

            return right;

        if(right == NULL)

            return left;

        return root;

    }

};

[面试题 04.09. 二叉搜索树序列](https://leetcode-cn.com/problems/bst-sequences-lcci/)

从左向右遍历一个数组，通过不断将其中的元素插入树中可以逐步地生成一棵二叉搜索树。给定一个由不同节点组成的二叉搜索树，输出所有可能生成此树的数组。

**示例：**  
给定如下二叉树

2

/ \

1 3

返回：

[

[2,1,3],

[2,3,1]

]

首先，可以把二叉树看作一个拓扑图，一棵树的“根结点的数值”总是先于它的“左右子树中的结点的数值”被插入树中。使用递归算法，依次枚举下一次插入的数值，变量buf用于构造数组。使用双端队列(变量q)存储这次可以插入的候选节点。

每次递归时，从双端队列q中取出候选值放入数组buf, 同时若候选值有左右子树，则将左右子树放进队列中。

若双端队列为q, 表示没有候选值，将buf放进结果ans中。

注意，每次选候选值结束时还原队列q与数组buf, 使其与输入时的值一致。

class Solution {

public:

    vector<vector<int>> BSTSequences(TreeNode\* root) {

        if (!root) return {{}};

        deque<TreeNode\*> q;

        q.push\_back(root);

        vector<int> buf;

        vector<vector<int> > ans;

        Inner(q, buf, ans);

        return ans;

    }

    void Inner(deque<TreeNode\*> &q, vector<int> &buf, vector<vector<int> > &ans) {

        if (q.empty()) {

            ans.push\_back(buf);

            return;

        }

        int size = q.size();

        while (size--) {

            TreeNode \*r = q.front(); q.pop\_front();

            buf.push\_back(r->val);

            int children = 0;

            if (r->left) {

                ++children;

                q.push\_back(r->left);

            }

            if (r->right) {

                ++children;

                q.push\_back(r->right);

            }

            Inner(q, buf, ans);

            while (children--) {

                q.pop\_back();

            }

            q.push\_back(r);

            buf.pop\_back();

        }

    }

};

[面试题 04.10. 检查子树](https://leetcode-cn.com/problems/check-subtree-lcci/)

检查子树。你有两棵非常大的二叉树：T1，有几万个节点；T2，有几万个节点。设计一个算法，判断 T2 是否为 T1 的子树。

如果 T1 有这么一个节点 n，其子树与 T2 一模一样，则 T2 为 T1 的子树，也就是说，从节点 n 处把树砍断，得到的树与 T2 完全相同。

**示例1:**

**输入**：t1 = [1, 2, 3], t2 = [2]

**输出**：true

**示例2:**

**输入**：t1 = [1, null, 2, 4], t2 = [3, 2]

**输出**：false

class Solution {

public:

    bool checkSubTree(TreeNode\* A, TreeNode\* B) {

    if(A == NULL || B == NULL)

        return false;

    return help(A, B) || checkSubTree(A->left, B) || checkSubTree(A->right, B);

    }

    bool help(TreeNode\* A, TreeNode\* B){

    if(A == NULL || B == NULL)

        return B == NULL;

    if(A->val != B->val)

        return false;

    return help(A->left, B->left) && help(A->right, B->right);

    }

};

[面试题 04.12. 求和路径](https://leetcode-cn.com/problems/paths-with-sum-lcci/)

给定一棵二叉树，其中每个节点都含有一个整数数值(该值或正或负)。设计一个算法，打印节点数值总和等于某个给定值的所有路径的数量。注意，路径不一定非得从二叉树的根节点或叶节点开始或结束，但是其方向必须向下(只能从父节点指向子节点方向)。

**示例:**  
给定如下二叉树，以及目标和 sum = 22，

5

/ \

4 8

/ / \

11 13 4

/ \ / \

7 2 5 1

返回:3 **解释：**和为 22 的路径有：[5,4,11,2], [5,8,4,5], [4,11,7]

class Solution {

public:

    int count = 0;

    void dfs(TreeNode\* root, int sum){

        if(root == NULL)

            return;

        sum -= root->val;

        if(sum == 0)

            count++;

        dfs(root->left, sum);

        dfs(root->right, sum);

    }

    int pathSum(TreeNode\* root, int sum) {

        if(root == NULL)

            return 0;

        dfs(root, sum);

        pathSum(root->left, sum);

        pathSum(root->right, sum);

        return count;

    }

};

[面试题 05.01. 插入](https://leetcode-cn.com/problems/insert-into-bits-lcci/)

插入。给定两个32位的整数N与M，以及表示比特位置的i与j。编写一种方法，将M插入N，使得M从N的第j位开始，到第i位结束。假定从j位到i位足以容纳M，也即若M = 10 011，那么j和i之间至少可容纳5个位。例如，不可能出现j = 3和i = 2的情况，因为第3位和第2位之间放不下M。

**示例1:输入**：N = 1024(10000000000), M = 19(10011), i = 2, j = 6

**输出**：N = 1100(10001001100)

**示例2:输入**： N = 0, M = 31(11111), i = 0, j = 4

**输出**：N = 31(11111)

class Solution {

public:

  int insertBits(int N, int M, int i, int j) {

    for (int k = i; k <= j; k++) {

      if (N & (1 << k))

        N -= (1 << k);

    }

    N += (M << i);

    return N;

  }

};

[面试题 05.02. 二进制数转字符串](https://leetcode-cn.com/problems/bianry-number-to-string-lcci/)

二进制数转字符串。给定一个介于0和1之间的实数（如0.72），类型为double，打印它的二进制表达式。如果该数字不在0和1之间，**或者**无法精确地用32位以内的二进制表示，则打印“ERROR”。

**示例1:**

**输入**：0.625

**输出**："0.101"

**示例2:**

**输入**：0.1

**输出**："ERROR"

**提示**：0.1无法被二进制准确表示

class Solution {

public:

    string printBin(double num) {

        string res = "0.";

        int i = 30; // 若 "0." 计为两位，i 初始化为 30，若计为 1 位，初始化为 31

        while (num > 0 && i--) {

            num \*= 2;

            if (num >= 1) {

                res.push\_back('1');

                --num;

            }

            else res.push\_back('0');

        }

        return num != 0 ? "ERROR" : res;

    }

};

[面试题 05.03. 翻转数位](https://leetcode-cn.com/problems/reverse-bits-lcci/)

给定一个32位整数 num，你可以将一个数位从0变为1。请编写一个程序，找出你能够获得的最长的一串1的长度。

**示例 1：**

**输入:** num = 1775(110111011112)

**输出:** 8

**示例 2：**

**输入:** num = 7(01112)

**输出:** 4

class Solution {

public:

    int reverseBits(int nums) {

        // 优化: 节省空间复杂度

        int maxVal = 0;

        // 表示左侧连续 1 的个数

        int pre = 0;

        int i = 0;

        unsigned int num = (unsigned int)nums;//加上这句就通过了，日

        while(i < 32){

            if((num & 1) == 0){

                int sum = 0;

                num >>= 1;

                i ++;

                while(i < 32 && num != 0 && (num & 1) == 1){

                    num >>= 1;

                    i ++;

                    sum ++;

                }

                // 注意: 当前字符从 0 变 1 后也要统计在内

                maxVal = max(maxVal, sum + pre + 1);

                // 当前右侧连续为 1 的个数 也是 下一个 0 的左侧连续为 1 的个数

                pre = sum;

            }else{

                pre ++;

                num >>= 1;

                i ++;

            }

        }

// 由于 2147483647 (0111 1111 1111 1111 1111 1111 1111 1111) 的存在, 所以必须跳出循环后再进行一次比较

        // 其次, 由于 -1 (1111 1111 1111 1111 1111 1111 1111 1111) 的存在, 所以必须保证值一定小于等于 32；其中 unsigned int a = -1，2^32 - 1 = 4294967295

        return max(maxVal, pre);

    }

};

[面试题 05.04. 下一个数](https://leetcode-cn.com/problems/closed-number-lcci/)

下一个数。给定一个正整数，找出与其二进制表达式中1的个数相同且大小最接近的那两个数（一个略大，一个略小）。

**示例1:**

**输入**：num = 2（或者0b10）

**输出**：[4, 1] 或者（[0b100, 0b1]）

**示例2:**

**输入**：num = 1

**输出**：[2, -1]

class Solution {

   public:

    vector<int> findClosedNumbers(int num) {

        bitset<32> small(num);

        bitset<32> bigger(num);//C++的 bitset 在 bitset 头文件中，它是一种类似数组的结构，它的每一个元素只能是０或１，每个元素仅用１bit空间。

        int s = -1;

        // small, 10 转 01，1移到左侧

        for (int i = 1; i < 32; i++) {

            if (small[i] == 1 && small[i - 1] == 0) {

                small.flip(i);

                small.flip(i - 1);

                for (int left = 0, right = i - 2; left < right;) {

                    while (left < right && small[left] == 0) left++;

                    while (left < right && small[right] == 1) right--;

                    small.flip(left);//flip()函数反置bitset中所有的位，即将1设为0，0设为1。如果指定pos，那么只有pos上的位被反置。

                    small.flip(right);

                }

                s = (int)small.to\_ulong();

                break;

            }

        }

        // bigger, 01转10，1移到最右侧

        int b = -1;

        for (int i = 1; i < 32; i++) {

            if (bigger[i] == 0 && bigger[i - 1] == 1) {

                bigger.flip(i);

                bigger.flip(i - 1);

                for (int left = 0, right = i - 2; left < right;) {

                    while (left < right && bigger[left] == 1) left++;

                    while (left < right && bigger[right] == 0) right--;

                    bigger.flip(left);

                    bigger.flip(right);

                }

                b = (int)bigger.to\_ulong();

                //控制bitset中的输出宽度的，输出与bitset<kuandu> ming(chushizhi)

                //to\_ulong()可以控制输出的最大限度，例如 kuandu是4，那么在输出时，每次结果都是 模除 2^4 - 1；如果宽度是5 那么结果都是 结果模除 2^5 - 1

                break;

            }

        }

        return {b, s};

    }

};

[面试题 05.06. 整数转换](https://leetcode-cn.com/problems/convert-integer-lcci/)

整数转换。编写一个函数，确定需要改变几个位才能将整数A转成整数B。

**示例1:**

**输入**：A = 29 （或者0b11101）, B = 15（或者0b01111）

**输出**：2

**示例2:**

**输入**：A = 1，B = 2

**输出**：2

// 不断对 c 进行移位操作，然后检查最低有效位。

class Solution {

public:

    int convertInteger(int A, int B) {

        int res = 0;

        for (unsigned c = A ^ B; c != 0; c = c >> 1)

            res += c & 1; // 数一数 c 中有几个 1

        return res;

    }

};

// 不断翻转最低有效位，计算需要多少次 c 会变成 0。其中 c = c & (c - 1) 是一个位操作的常用问题，可以特别注意一下。

class Solution {

public:

    int convertInteger(int A, int B) {

        int res = 0;

        for (unsigned c = A ^ B; c != 0; c = c & (c - 1))

            res ++;

        return res;

    }

};

// 时间复杂度：O(1)。

// 空间复杂度：O(1)。

[面试题 05.07. 配对交换](https://leetcode-cn.com/problems/exchange-lcci/)

配对交换。编写程序，交换某个整数的奇数位和偶数位，尽量使用较少的指令（也就是说，位0与位1交换，位2与位3交换，以此类推）。

**示例1:**

**输入**：num = 2（或者0b10）

**输出** 1 (或者 0b01)

**示例2:**

**输入**：num = 3

**输出**：3

如果一对一对地交换，必定有很多困难。

因此可以先操作奇数位，再操作偶数位。

对于奇数位，使用 101010（即 0xAA）作为掩码，提取奇数位，并把它们右移一位；

对于偶数位，使用 010101（即 0x55）作为掩码，提取偶数位，并把它们左移一位。

最后，将两次操作的结果合并成一个值。

class Solution {

public:

    int exchangeBits(int num) {

        int even = (num & 0xaaaaaaaa) >> 1;

        int odd = (num & 0x55555555) << 1;

        return even | odd;

    }

};

[面试题 05.08. 绘制直线](https://leetcode-cn.com/problems/draw-line-lcci/)

绘制直线。有个单色屏幕存储在一个一维数组中，使得32个连续像素可以存放在一个 int 里。屏幕宽度为w，且w可被32整除（即一个 int 不会分布在两行上），屏幕高度可由数组长度及屏幕宽度推算得出。请实现一个函数，绘制从点(x1, y)到点(x2, y)的水平线。

给出数组的长度 length，宽度 w（以比特为单位）、直线开始位置 x1（比特为单位）、直线结束位置 x2（比特为单位）、直线所在行数 y。返回绘制过后的数组。

**示例1:**

**输入**：length = 1, w = 32, x1 = 30, x2 = 31, y = 0

**输出**：[3]

**说明**：在第0行的第30位到第31为画一条直线，屏幕表示为[0b000000000000000000000000000000011]

**示例2:**

**输入**：length = 3, w = 96, x1 = 0, x2 = 95, y = 0

**输出**：[-1, -1, -1]

class Solution {

public:

    vector<int> drawLine(int length, int w, int x1, int x2, int y) {

        vector<int> p(length,0);

        if (length == 0) return p;

        int row = w / 32;       //一行有多少int

        int start = row \* y + x1 / 32;  //线段头所在的int

        int start\_s = x1 % 32;          //线段头所在的int的左边需要空出0的个数

        int end = row \* y + x2 / 32;    //线段尾所在的int

        int end\_s = 31 - x2 % 32;       //线段尾所在的int的右边需要空出0的个数

        for (int i = start; i <= end; i++) {

            p[i] = 0xffffffff;          //置为-1；

        }

        if (start == end) { //对头尾在一个int里时单独处理

            p[start] = (unsigned int)p[start] >> start\_s+ end\_s;

            p[end] = (unsigned int)p[end] << end\_s;

        }

        else {

            p[start] = (unsigned int)p[start] >> start\_s;

            p[end] = (unsigned int)p[end] << end\_s;

        }

        return p;

    }

};

[面试题 08.01. 三步问题](https://leetcode-cn.com/problems/three-steps-problem-lcci/)

三步问题。有个小孩正在上楼梯，楼梯有n阶台阶，小孩一次可以上1阶、2阶或3阶。实现一种方法，计算小孩有多少种上楼梯的方式。结果可能很大，你需要对结果模1000000007。

**示例1:**

**输入**：n = 3

**输出**：4

**说明**: 有四种走法

**示例2:**

**输入**：n = 5

**输出**：13

class Solution {

public:

    int waysToStep(int n){

        if(n <= 2) return n;

        if (n == 3) return 4;

        int a = 1,b=2,c=4,d;

        for (int i = 4; i <= n; i++){

            d = ((a+b)%1000000007+c)% 1000000007;

            a = b;

            b = c;

            c = d;

        }

        return d;

    }

};

[面试题 08.02. 迷路的机器人](https://leetcode-cn.com/problems/robot-in-a-grid-lcci/)

设想有个机器人坐在一个网格的左上角，网格 r 行 c 列。机器人只能向下或向右移动，但不能走到一些被禁止的网格（有障碍物）。设计一种算法，寻找机器人从左上角移动到右下角的路径。



网格中的障碍物和空位置分别用 1 和 0 来表示。

返回一条可行的路径，路径由经过的网格的行号和列号组成。左上角为 0 行 0 列。如果没有可行的路径，返回空数组。

**示例 1:输入:**

[

  [**0**,**0**,**0**],

  [0,1,**0**],

  [0,0,**0**]

]

**输出:** [[0,0],[0,1],[0,2],[1,2],[2,2]]

**解释:** 输入中标粗的位置即为输出表示的路径，即

0行0列（左上角） -> 0行1列 -> 0行2列 -> 1行2列 -> 2行2列（右下角）

class Solution {

public:

    bool dfs(vector<vector<int>>& g, int x, int y, vector<vector<int>>& path) {

        int m = g.size();

        int n = g[0].size();

        if (g[x][y] == 1) return false;

        if (x ==  m - 1 && y == n - 1) {

            path.push\_back({x, y});

            return true;

        }

        path.push\_back({x, y});

        if (x + 1 < m && dfs(g, x + 1, y, path)) {

            return true;

        }

        if (y + 1 < n && dfs(g, x, y + 1, path)) {

            return true;

        }

        path.pop\_back();

        g[x][y] = 1; //不可达

        return false;

    }

    vector<vector<int>> pathWithObstacles(vector<vector<int>>& obstacleGrid) {

        vector<vector<int>> path;

        dfs(obstacleGrid, 0, 0, path);

        return path;

    }

};

[面试题 08.03. 魔术索引](https://leetcode-cn.com/problems/magic-index-lcci/)

魔术索引。 在数组A[0...n-1]中，有所谓的魔术索引，满足条件A[i] = i。给定一个有序整数数组，编写一种方法找出魔术索引，若有的话，在数组A中找出一个魔术索引，如果没有，则返回-1。若有多个魔术索引，返回索引值最小的一个。

**示例1:**

**输入**：nums = [0, 2, 3, 4, 5]

**输出**：0

**说明**: 0下标的元素为0

**示例2:**

**输入**：nums = [1, 1, 1]

**输出**：1

class Solution {

public:

    int findMagicIndex(vector<int>& nums) {

        int i=0;

        while(i < nums.size())

        {

            if(i == nums[i])

                return i;

            i = max(i + 1, nums[i]);

        }

        return -1;

    }

};

[面试题 08.04. 幂集](https://leetcode-cn.com/problems/power-set-lcci/)

幂集。编写一种方法，返回某集合的所有子集。集合中不包含重复的元素。

说明：解集不能包含重复的子集。

**示例:输入**： nums = [1,2,3]

**输出**：

[

[3],

  [1],

  [2],

  [1,2,3],

  [1,3],

  [2,3],

  [1,2],

  []

]

先在大容器里面存一个空容器[].

第一轮：n = 1

大容器：[] | [1];

第二轮：n = 2

大容器：[] [1] | [2] [1,2];

第三轮：n = 3

大容器：[] [1] [2] [1,2] | [3] [1,3] [2,3] [1,2,3] ;

class Solution {

public:

 vector<vector<int>> v;

    vector<vector<int>> subsets(vector<int>& nums) {

        v.push\_back({});

        for(int i = 0 ; i < nums.size() ; i++){

             vector<vector<int>> b = v;

            for(int j = 0 ; j < b.size() ; j++){

                vector<int> a = v[j];

                a.push\_back(nums[i]);

                v.push\_back(a);

            }

        }

        return v;

    }

};

[面试题 08.05. 递归乘法](https://leetcode-cn.com/problems/recursive-mulitply-lcci/)

递归乘法。 写一个递归函数，不使用 \* 运算符， 实现两个正整数的相乘。可以使用加号、减号、位移，但要吝啬一些。

**示例1:**

**输入**：A = 1, B = 10

**输出**：10

**示例2:**

**输入**：A = 3, B = 4

**输出**：12

class Solution {

public:

    int multiply(int A, int B) {

        if (A > B) return multiply(B, A);

        if (A == 0) return 0;

        return B + multiply(A - 1, B);

    }

};

[面试题 08.06. 汉诺塔问题](https://leetcode-cn.com/problems/hanota-lcci/)

在经典汉诺塔问题中，有 3 根柱子及 N 个不同大小的穿孔圆盘，盘子可以滑入任意一根柱子。一开始，所有盘子自上而下按升序依次套在第一根柱子上(即每一个盘子只能放在更大的盘子上面)。移动圆盘时受到以下限制:  
(1) 每次只能移动一个盘子;  
(2) 盘子只能从柱子顶端滑出移到下一根柱子;  
(3) 盘子只能叠在比它大的盘子上。

请编写程序，用栈将所有盘子从第一根柱子移到最后一根柱子。

你需要原地修改栈。

**示例1:**

**输入**：A = [2, 1, 0], B = [], C = []

**输出**：C = [2, 1, 0]

**示例2:**

**输入**：A = [1, 0], B = [], C = []

**输出**：C = [1, 0]

class Solution {

public:

    void hanota(vector<int>& A, vector<int>& B, vector<int>& C) {

        int n = A.size();

        move(n, A, B, C);

    }

    void move(int n, vector<int>& A, vector<int>& B, vector<int>& C){

        if (n == 1){

            C.push\_back(A.back());

            A.pop\_back();

            return;

        }

        move(n-1, A, C, B);    // 将A上面n-1个通过C移到B

        C.push\_back(A.back());  // 将A最后一个移到C

        A.pop\_back();          // 这时，A空了

        move(n-1, B, A, C);     // 将B上面n-1个通过空的A移到C

    }

};

[面试题 08.07. 无重复字符串的排列组合](https://leetcode-cn.com/problems/permutation-i-lcci/)

无重复字符串的排列组合。编写一种方法，计算某字符串的所有排列组合，字符串每个字符均不相同。

**示例1:**

**输入**：S = "qwe"

**输出**：["qwe", "qew", "wqe", "weq", "ewq", "eqw"]

**示例2:**

**输入**：S = "ab"

**输出**：["ab", "ba"]

class Solution {

public:

    vector<string> ans;

    vector<string> permutation(string S) {

        dfs(S, 0);

        return ans;

    }

void dfs(string S, int idx) {

        int n = S.size();

        if (idx == n) ans.emplace\_back(S);

        for (int i = idx; i != n; ++i) {

            string tmp = S;

            swap(tmp[idx], tmp[i]);

            dfs(tmp, idx + 1);

        }

    }

};

[面试题 08.08. 有重复字符串的排列组合](https://leetcode-cn.com/problems/permutation-ii-lcci/)

有重复字符串的排列组合。编写一种方法，计算某字符串的所有排列组合。

**示例1:**

**输入**：S = "qqe"

**输出**：["eqq","qeq","qqe"]

**示例2:**

**输入**：S = "ab"

**输出**：["ab", "ba"]

class Solution {

public:

    vector<string>res;

    vector<string> permutation(string S) {

        string s;

        int n=S.size();

        sort(S.begin(),S.end());//排序使得重复字符相邻

        vector<int>flag(n);

        process(S,s,n,flag);

        return res;

    }

    void process(string S,string& s,int n,vector<int>&flag)

    {

        //终止条件

        if(s.size()==S.size()){

            res.push\_back(s);

        }

        else{

            for(int i=0;i<n;i++){

                if(flag[i]==0){

                    if(i>0&&S[i]==S[i-1]&&flag[i-1]==1)continue;//若有重复字符，跳过该组合

                    s+=S[i];

                    flag[i]=1;

                    process(S,s,n,flag);

                    s.pop\_back();//回溯到上一步

                    flag[i]=0;

                }

            }

        }

    }

};

[面试题 08.09. 括号](https://leetcode-cn.com/problems/bracket-lcci/)

括号。设计一种算法，打印n对括号的所有合法的（例如，开闭一一对应）组合。

说明：解集不能包含重复的子集。

例如，给出 n = 3，生成结果为：

[

"((()))",

"(()())",

"(())()",

"()(())",

"()()()"

]

class Solution {

    vector<string> ans;

public:

    vector<string> generateParenthesis(int n) {

        help(n, n, "");

        return ans;

    }

    void help(int l, int r, string a) {

        if (l == 0 && r == 0) {

            ans.emplace\_back(a);

            return;

        }

        if (l != 0) help(l - 1, r, a + '(');

        if (l < r) help(l, r - 1, a + ')');

    }

};

[面试题 08.10. 颜色填充](https://leetcode-cn.com/problems/color-fill-lcci/)

编写函数，实现许多图片编辑软件都支持的「颜色填充」功能。

待填充的图像用二维数组 image 表示，元素为初始颜色值。初始坐标点的横坐标为 sr 纵坐标为 sc。需要填充的新颜色为 newColor 。

「周围区域」是指颜色相同且在上、下、左、右四个方向上存在相连情况的若干元素。

请用新颜色填充初始坐标点的周围区域，并返回填充后的图像。

**示例：输入**：

image = [[1,1,1],[1,1,0],[1,0,1]] sr = 1, sc = 1, newColor = 2

**输出**：[[2,2,2],[2,2,0],[2,0,1]]

**解释**:

初始坐标点位于图像的正中间，坐标 (sr,sc)=(1,1) 。

初始坐标点周围区域上所有符合条件的像素点的颜色都被更改成 2 。

注意，右下角的像素没有更改为 2 ，因为它不属于初始坐标点的周围区域。

class Solution {

public:

    vector<vector<int>> floodFill(vector<vector<int>>& image, int sr, int sc, int newColor) {

        int oldColor = image[sr][sc];   //记录旧坐标的像素

        if (image[sr][sc] == newColor) return image; //颜色相同则无需修改

        image[sr][sc] = newColor;   //染色

        int dx[4] = {-1, 0, 1, 0}, dy[4] = {0, 1, 0, -1}; //上右下左

        for (int i = 0; i < 4; i ++) {

            int x = sr + dx[i], y = sc + dy[i];

            if (x >=0 && y >= 0 && x < image.size() && y < image[0].size() && image[x][y] == oldColor)

                floodFill (image, x, y, newColor);

        }

        return image;

    }

};

[面试题 08.11. 硬币](https://leetcode-cn.com/problems/coin-lcci/)

硬币。给定数量不限的硬币，币值为25分、10分、5分和1分，编写代码计算n分有几种表示法。(结果可能会很大，你需要将结果模上1000000007)

**示例1:**

**输入**: n = 5

**输出**：2

**解释**: 有两种方式可以凑成总金额:

5=5

5=1+1+1+1+1

**示例2:**

**输入**: n = 10

**输出**：4

**解释**: 有四种方式可以凑成总金额:

10=10

10=5+5

10=5+1+1+1+1+1

10=1+1+1+1+1+1+1+1+1+1

class Solution {

private:

    static constexpr int mod = 1000000007;

    static constexpr int coins[4] = {25, 10, 5, 1};

public:

    int waysToChange(int n) {

        vector<int> f(n + 1);

        f[0] = 1;

        for (int c = 0; c < 4; ++c) {

            int coin = coins[c];

            for (int i = coin; i <= n; ++i) {

                f[i] = (f[i] + f[i - coin]) % mod;

            }

        }

        return f[n];

    }

};

[面试题 08.12. 八皇后](https://leetcode-cn.com/problems/eight-queens-lcci/)

设计一种算法，打印 N 皇后在 N × N 棋盘上的各种摆法，其中每个皇后都不同行、不同列，也不在对角线上。这里的“对角线”指的是所有的对角线，不只是平分整个棋盘的那两条对角线。

**注意：**本题相对原题做了扩展

**示例:**

**输入**：4

**输出**：[[".Q..","...Q","Q...","..Q."],["..Q.","Q...","...Q",".Q.."]]

**解释**: 4 皇后问题存在如下两个不同的解法。

[

 [".Q..",  // 解法 1

  "...Q",

  "Q...",

  "..Q."],

 ["..Q.",  // 解法 2

  "Q...",

  "...Q",

  ".Q.."]

]

function backTrack(供选择的列表，路径){

    if(终止条件){

        退出;

    }

    做选择;  //从列表选择一项加入路径

    backTrack(列表，路径);

    撤销选择;

}

class Solution {

public:

    vector<vector<string>> solveNQueens(int n) {

        auto solutions = vector<vector<string>>();

        auto queens = vector<int>(n, -1);

        auto columns = unordered\_set<int>();

        auto diagonals1 = unordered\_set<int>();

        auto diagonals2 = unordered\_set<int>();

        backtrack(solutions, queens, n, 0, columns, diagonals1, diagonals2);

        return solutions;

    }

    void backtrack(vector<vector<string>> &solutions, vector<int> &queens, int n, int row, unordered\_set<int> &columns, unordered\_set<int> &diagonals1, unordered\_set<int> &diagonals2) {

        if (row == n) {

            vector<string> board = generateBoard(queens, n);

            solutions.push\_back(board);

        } else {

            for (int i = 0; i < n; i++) {

                if (columns.find(i) != columns.end()) {

                    continue;

                }

                int diagonal1 = row - i;

                if (diagonals1.find(diagonal1) != diagonals1.end()) {

                    continue;

                }

                int diagonal2 = row + i;

                if (diagonals2.find(diagonal2) != diagonals2.end()) {

                    continue;

                }

                queens[row] = i;

                columns.insert(i);

                diagonals1.insert(diagonal1);

                diagonals2.insert(diagonal2);

                backtrack(solutions, queens, n, row + 1, columns, diagonals1, diagonals2);

                queens[row] = -1;

                columns.erase(i);

                diagonals1.erase(diagonal1);

                diagonals2.erase(diagonal2);

            }

        }

    }

    vector<string> generateBoard(vector<int> &queens, int n) {

        auto board = vector<string>();

        for (int i = 0; i < n; i++) {

            string row = string(n, '.');

            row[queens[i]] = 'Q';

            board.push\_back(row);

        }

        return board;

    }

};

[面试题 08.13. 堆箱子](https://leetcode-cn.com/problems/pile-box-lcci/)

难度困难22收藏分享切换为英文关注反馈

堆箱子。给你一堆n个箱子，箱子宽 wi、深 di、高 hi。箱子不能翻转，将箱子堆起来时，下面箱子的宽度、高度和深度必须大于上面的箱子。实现一种方法，搭出最高的一堆箱子。箱堆的高度为每个箱子高度的总和。

输入使用数组[wi, di, hi]表示每个箱子。

**示例1:**

**输入**：box = [[1, 1, 1], [2, 2, 2], [3, 3, 3]]

**输出**：6

**示例2:**

**输入**：box = [[1, 1, 1], [2, 3, 4], [2, 6, 7], [3, 4, 5]]

**输出**：10

class Solution {

public:

    int pileBox(vector<vector<int>>& box) {

        sort(box.begin(), box.end(), [](const vector<int>& a, const vector<int>& b) {return a[0]<b[0];});

        vector<int> dp(box.size(),0);

        dp[0]=box[0][2];

        int ans=dp[0];

        for(int i=1;i<box.size();i++){

            int maxh=0;//必须初始化为0

            for(int j=0;j<i;j++)

                if(box[j][0]<box[i][0]&&box[j][1]<box[i][1]&&box[j][2]<box[i][2])

                    maxh=max(maxh,dp[j]);

            dp[i]=maxh+box[i][2];

            ans=max(ans,dp[i]);

        }

        return ans;

    }

};

300. 最长上升子序列

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(n2) 。

进阶: 你能将算法的时间复杂度降低到 O(n log n) 吗?

class Solution {

public:

    int lengthOfLIS(vector<int>& nums) {

        int n=(int)nums.size();

        if (n == 0) return 0;

        vector<int> dp(n, 0);

        for (int i = 0; i < n; ++i) {

            dp[i] = 1;

            for (int j = 0; j < i; ++j) {

                if (nums[j] < nums[i]) {

                    dp[i] = max(dp[i], dp[j] + 1);

                }

            }

        }

        return \*max\_element(dp.begin(), dp.end());

    }

};

// 时间复杂度：O(n^2)，其中 n为数组nums 的长度。动态规划的状态数为n，计算状态 dp[i]时，需要O(n)的时间遍历dp[0…i−1] 的所有状态，所以总时间复杂度为 O(n^2)

// 空间复杂度：O(n)，需要额外使用长度为 n 的dp数组。

class Solution {

public:

    int lengthOfLIS(vector<int>& nums) {

        int len = 1, n = (int)nums.size();

        if (n == 0) return 0;

        vector<int> d(n + 1, 0);

        d[len] = nums[0];

        for (int i = 1; i < n; ++i) {

            if (nums[i] > d[len]) d[++len] = nums[i];

            else{

                int l = 1, r = len, pos = 0; // 如果找不到说明所有的数都比 nums[i] 大，此时要更新 d[1]，所以这里将 pos 设为 0

                while (l <= r) {

                    int mid = (l + r) >> 1;

                    if (d[mid] < nums[i]) {

                        pos = mid;

                        l = mid + 1;

                    }

                    else r = mid - 1;

                }

                d[pos + 1] = nums[i];

            }

        }

        return len;

    }

};

无序列表最关键的一句在于： 数组 d[i]表示长度为 i 的最长上升子序列的末尾元素的最小值，即在数组 1,2,3,4,5,6中长度为3的上升子序列可以为 1,2,3也可以为 2,3,4等等但是d[3]=3，即子序列末尾元素最小为3。

无序列表解释清了数组d的含义之后，我们接着需要证明数组d具有单调性，即证明i<j时，d[i]<d[j]，使用反证法，假设存在k<j时，d[k]>d[j]，但在长度为j，末尾元素为d[j]的子序列A中，将后j-i个元素减掉，可以得到一个长度为i的子序列B，其末尾元素t1必然小于d[j]（因为在子序列A中，t1的位置上在d[j]的后面），而我们假设数组d必须符合表示长度为 i 的最长上升子序列的末尾元素的最小值，此时长度为i的子序列的末尾元素t1<d[j]<d[k]，即t1<d[k]，所以d[k]不是最小的，与题设相矛盾，因此可以证明其单调性

无序列表证明单调性有两个好处：1.可以使用二分法；2.数组d的长度即为最长子序列的长度；

// 时间复杂度：O(nlogn)。数组nums 的长度为n，我们依次用数组中的元素去更新d数组，而更新d数组时需要进行O(logn)的二分搜索，所以总时间复杂度为O(nlogn)。

// 空间复杂度：O(n)，需要额外使用长度为n的d数组。

[面试题 08.14. 布尔运算](https://leetcode-cn.com/problems/boolean-evaluation-lcci/)

给定一个布尔表达式和一个期望的布尔结果 result，布尔表达式由 0 (false)、1 (true)、& (AND)、 | (OR) 和 ^ (XOR) 符号组成。实现一个函数，算出有几种可使该表达式得出 result 值的括号方法。

**示例 1:**

**输入:** s = "1^0|0|1", result = 0

**输出:** 2

**解释:** 两种可能的括号方法是

1^(0|(0|1))

1^((0|0)|1)

**示例 2:**

**输入:** s = "0&0&0&1^1|0", result = 1

**输出:** 10

记：F[i][j][a=0/1]表示第i到j个数字计算结果为a的方案数。

转移：枚举区间分割点，讨论左右区间计算结果，方案数增量为左右方案数相乘。

时间复杂度：O(n^3) 空间复杂度：O(n^2)

思路：在某个运算符处进行分割，然后因为运算符左右两边只存在两种结果。 0 和 1

比如 "0&0&0     &   1^1|0"

运算符 & 分割后，左边的结果最终只能是 1 或 0 其中一个

右边的结果最终只能是 1 或 0 其中一个，即最终是 0 和 0 、1 和 1、 0 和 1、 1 和 0 进行 &，那么我们只需要统计左右两边存在多少个 0 和 1，进行乘积即可

class Solution {

public:

    int countEval(string s, int result) {

        int len=s.size();

        vector<int> A;

        vector<char> opt;

        for (int i=0;i<len;++i)

            if (i&1) opt.push\_back(s[i]);

            else A.push\_back(s[i]-'0');

        int n=A.size();

        int F[n][n][2];

        for (int i=0;i<n;++i) F[i][i][A[i]]=1,F[i][i][A[i]^1]=0;

        for (int l=2;l<=n;++l)

        {

            for (int i=0;i+l-1<n;++i)

            {

                int j=i+l-1;

                F[i][j][0]=F[i][j][1]=0;

                for (int k=i;k<j;++k)

                    if (opt[k]=='&')

                    {

                        F[i][j][1]+=F[i][k][1]\*F[k+1][j][1];

                        F[i][j][0]+=F[i][k][1]\*F[k+1][j][0];

                        F[i][j][0]+=F[i][k][0]\*F[k+1][j][1];

                        F[i][j][0]+=F[i][k][0]\*F[k+1][j][0];

                    }

                    else if (opt[k]=='|')

                    {

                        F[i][j][1]+=F[i][k][1]\*F[k+1][j][1];

                        F[i][j][1]+=F[i][k][1]\*F[k+1][j][0];

                        F[i][j][1]+=F[i][k][0]\*F[k+1][j][1];

                        F[i][j][0]+=F[i][k][0]\*F[k+1][j][0];

                    }

                    else

                    {

                        F[i][j][0]+=F[i][k][1]\*F[k+1][j][1];

                        F[i][j][1]+=F[i][k][1]\*F[k+1][j][0];

                        F[i][j][1]+=F[i][k][0]\*F[k+1][j][1];

                        F[i][j][0]+=F[i][k][0]\*F[k+1][j][0];

                    }

            }

        }

        return F[0][n-1][result];

    }

};

[面试题 10.01. 合并排序的数组](https://leetcode-cn.com/problems/sorted-merge-lcci/)

给定两个排序后的数组 A 和 B，其中 A 的末端有足够的缓冲空间容纳 B。 编写一个方法，将 B 合并入 A 并排序。

初始化 A 和 B 的元素数量分别为 *m* 和 *n*。

**示例:**

**输入:**

A = [1,2,3,0,0,0], m = 3

B = [2,5,6], n = 3

**输出:** [1,2,2,3,5,6]

双指针，时间复杂度：O(m+n)，空间复杂度：O(m+n)

class Solution {

public:

    void merge(vector<int>& A, int m, vector<int>& B, int n) {

        int pa = 0, pb = 0;

        int sorted[m + n];

        int cur;

        while (pa < m || pb < n) {

            if (pa == m)

                cur = B[pb++];

            else if (pb == n)

                cur = A[pa++];

            else if (A[pa] < B[pb])

                cur = A[pa++];

            else

                cur = B[pb++];

            sorted[pa + pb - 1] = cur;

        }

        for (int i = 0; i != m + n; ++i)

            A[i] = sorted[i];

    }

};

逆向双指针

时间复杂度：O(m+n)，空间复杂度：O(1)，直接对数组 A 原地修改，不需要额外空间。

class Solution {

public:

    void merge(vector<int>& A, int m, vector<int>& B, int n) {

        int pa = m - 1, pb = n - 1;

        int tail = m + n - 1;

        int cur;

        while (pa >= 0 || pb >= 0) {

            if (pa == -1)

                cur = B[pb--];

            else if (pb == -1)

                cur = A[pa--];

            else if (A[pa] > B[pb])

                cur = A[pa--];

            else

                cur = B[pb--];

            A[tail--] = cur;

        }

    }

};

[面试题 10.02. 变位词组](https://leetcode-cn.com/problems/group-anagrams-lcci/)

编写一种方法，对字符串数组进行排序，将所有变位词组合在一起。变位词是指字母相同，但排列不同的字符串。

**注意：**本题相对原题稍作修改

**示例:**

**输入:** ["eat", "tea", "tan", "ate", "nat", "bat"],

**输出:**

[

["ate","eat","tea"],

["nat","tan"],

["bat"]

]

class Solution {

public:

    vector<vector<string>> groupAnagrams(vector<string>& strs) {

          vector<vector<string>>vec;

          unordered\_map<string,vector<int>>map;

          for(int i=0;i<strs.size();i++){

              string tmp=strs[i];

              sort(tmp.begin(),tmp.end());

              map[tmp].push\_back(i);

          }

          for(auto it:map){

              auto index=it.second;

              vector<string>res;

              for(auto num:index){

                  res.push\_back(strs[num]);

              }

              vec.push\_back(res);

          }

        return vec;

    }

};

[面试题 10.03. 搜索旋转数组](https://leetcode-cn.com/problems/search-rotate-array-lcci/)

搜索旋转数组。给定一个排序后的数组，包含n个整数，但这个数组已被旋转过很多次了，次数不详。请编写代码找出数组中的某个元素，假设数组元素原先是按升序排列的。若有多个相同元素，返回索引值最小的一个。

**示例1:**

**输入**: arr = [15, 16, 19, 20, 25, 1, 3, 4, 5, 7, 10, 14], target = 5

**输出**: 8（元素5在该数组中的索引）

**示例2:**

**输入**：arr = [15, 16, 19, 20, 25, 1, 3, 4, 5, 7, 10, 14], target = 11

**输出**：-1 （没有找到）

对于能判断升序区间的情况，根据目标值的大小移动边界。

对于不能判断升序区间的情况，需要逐步清除重复值。

没有重复值的最优情况时间复杂度是O(log N)，全部或几乎全部是重复值的最差情况时间复杂度是O(N)。

                                    nums[left] <= target

                              ┌─  && target <= nums[mid]   ──>  right = mid

                              │   （目标在左边的升序区间中）         （右边界移动到mid）

  ┌─  nums[left] < nums[mid] ─┼

  │     （左边区间升序）         │

  │                           └─    否则目标在右半边          ──>  left = mid + 1

  │                                                             （左边界移动到mid+1）

  │

  │                                 nums[left] <= target

  │                           ┌─  || target <= nums[mid]   ──>  right = mid

  │                           │    （目标在左半边）                （右边界移动到mid）

 ─┼─  nums[left] > nums[mid] ─┼

  │     （左边不是升序）         │

  │                           └─    否则目标在右半边          ──>  left = mid + 1

  │                                                              （左边界移动到mid+1）

  │

  │

  │                           ┌─   nums[left] != target    ──>  left++

  │                           │     （左值不等于目标               （需要逐一清理重复值）

  └─ nums[left] == nums[mid] ─┼         说明还没找到）

      （可能是已经找到了目标      │

        也可能是遇到了重复值）    └─   nums[left] == target    ──>  right = left

                                    （左值等于目标                 （将右边界移动到left，循环结束）

                                      已经找到最左边的目标值）

class Solution {

public:

    int search(vector<int>& nums, int target) {

        int left = 0;

        int right = nums.size() - 1;

        if (right == -1)

            return -1;

        while (left < right) {                                         // 循环结束条件left==right

            int mid = left + (right - left) / 2;

            if (nums[left] < nums[mid]) {                              // 如果左值小于中值，说明左边区间升序

                if (nums[left] <= target && target <= nums[mid]) {     // 如果目标在左边的升序区间中，右边界移动到mid

                    right = mid;

                } else {                                               // 否则目标在右半边，左边界移动到mid+1

                    left = mid + 1;

                }

            } else if (nums[left] > nums[mid]) {                       // 如果左值大于中值，说明左边不是升序，右半边升序

                if (nums[left] <= target || target <= nums[mid]) {     // 如果目标在左边，右边界移动到mid

                    right = mid;

                } else {                                               // 否则目标在右半边，左边界移动到mid+1

                    left = mid + 1;

                }

            } else if (nums[left] == nums[mid]) {                      // 如果左值等于中值，可能是已经找到了目标，也可能是遇到了重复值

                if (nums[left] != target) {                            // 如果左值不等于目标，说明还没找到，需要逐一清理重复值。

                    left++;

                } else {                                               // 如果左值等于目标，说明已经找到最左边的目标值

                    right = left;                                      // 将右边界移动到left，循环结束

                }

            }

        }

        return (nums[left] == target) ? left : -1;                     // 返回left，或者-1

    }

};

[面试题 10.05. 稀疏数组搜索](https://leetcode-cn.com/problems/sparse-array-search-lcci/)

稀疏数组搜索。有个排好序的字符串数组，其中散布着一些空字符串，编写一种方法，找出给定字符串的位置。

**示例1:**

**输入**: words = ["at", "", "", "", "ball", "", "", "car", "", "","dad", "", ""], s = "ta"

**输出**：-1

**说明**: 不存在返回-1。

**示例2:**

**输入**：words = ["at", "", "", "", "ball", "", "", "car", "", "","dad", "", ""], s = "ball"

**输出**：4

class Solution {

public:

    int findString(vector<string>& words, string s) {

        int left = 0, right = words.size() - 1;

        while (left <= right) {

            if (words[left].size() == 0) {

                left++;

                continue;

            }

            if (words[right].size() == 0) {

                right--;

                continue;

            }

            int mid = (right + left) / 2;

            while (words[mid].size() == 0) {

                mid++;

                if (mid == right) {

                    right = (right + left) / 2;

                    continue;

                }

            }

            if (words[mid] == s)

                return mid;

            else if (words[mid] > s) {

                right = mid - 1;

            }

            else {

                left = mid + 1;

            }

        }

        return -1;

    }

};

[面试题 10.09. 排序矩阵查找](https://leetcode-cn.com/problems/sorted-matrix-search-lcci/)

难度中等11收藏分享切换为英文关注反馈

给定M×N矩阵，每一行、每一列都按升序排列，请编写代码找出某元素。

**示例:**

现有矩阵 matrix 如下：

[

[1, 4, 7, 11, 15],

[2, 5, 8, 12, 19],

[3, 6, 9, 16, 22],

[10, 13, 14, 17, 24],

[18, 21, 23, 26, 30]

]

给定 target = 5，返回 true。

给定 target = 20，返回 false。

class Solution {

public:

    bool searchMatrix(vector<vector<int>>& matrix, int target) {

        int i = matrix.size() - 1;

        int j = 0;

        while(i >= 0 && j < matrix[0].size()){

            if(matrix[i][j] == target)

                return true;

            if(matrix[i][j] > target)

                i--;

            else

                j++;

        }

        return false;

    }

};

[面试题 10.10. 数字流的秩](https://leetcode-cn.com/problems/rank-from-stream-lcci/)

假设你正在读取一串整数。每隔一段时间，你希望能找出数字 x 的秩(小于或等于 x 的值的个数)。请实现数据结构和算法来支持这些操作，也就是说：

实现 track(int x) 方法，每读入一个数字都会调用该方法；

实现 getRankOfNumber(int x) 方法，返回小于或等于 x 的值的个数。

**注意：**本题相对原题稍作改动

**示例:输入:**

["StreamRank", "getRankOfNumber", "track", "getRankOfNumber"]

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

**输出:** null,0,null,1]

class StreamRank

{

    struct TreeNode

    {

        int sum;

        int val;

        TreeNode \*left;

        TreeNode \*right;

        TreeNode(int x) : val(x), sum(1), left(NULL), right(NULL) {}

    };

public:

    StreamRank()

    {

        root = NULL;

    }

    void track(int x)

    {

        if (!root)

        {

            root = new TreeNode(x);

            return;

        }

        TreeNode \*cur = root;

        while (cur)

        {

            if (x <= cur->val)

            {

                cur->sum++;

                //相同的数字只增加其计数

                if (x == cur->val)

                    break;

                //如果左结点为空，则可以插入，结束搜索

                if (!cur->left)

                {

                    cur->left = new TreeNode(x);

                    break;

                }

                //否则计数向左搜索

                cur = cur->left;

            }

            else

            {

                if (!cur->right)

                {

                    cur->right = new TreeNode(x);

                    break;

                }

                //否则继续向右搜索

                cur = cur->right;

            }

        }

    }

    int getRankOfNumber(int x)

    {

        int ret = 0;

        if (root != NULL)

        {

            TreeNode \*tmp = root;

            while (tmp && tmp->val != x)

            {

                if (x < tmp->val)

                    tmp = tmp->left;

                else //往右边查找的时候，加上父结点的计数

                {

                    ret += tmp->sum;

                    tmp = tmp->right;

                }

            }

            if (tmp != NULL)

                ret = ret + tmp->sum;

        }

        return ret;

    }

private:

    TreeNode \*root;

};

[面试题 10.11. 峰与谷](https://leetcode-cn.com/problems/peaks-and-valleys-lcci/)

在一个整数数组中，“峰”是大于或等于相邻整数的元素，相应地，“谷”是小于或等于相邻整数的元素。例如，在数组{5, 8, 6, 2, 3, 4, 6}中，{8, 6}是峰， {5, 2}是谷。现在给定一个整数数组，将该数组按峰与谷的交替顺序排序。

**示例:**

**输入:** [5, 3, 1, 2, 3]

**输出:** [5, 1, 3, 2, 3]

假设按照峰-谷-峰的顺序排列数组，那么遍历一遍数组：

（1）如果i为峰的位置，则判断当前位置是否小于前一个位置（前一个为谷），若小于，则交换，大于则不处理。即： if(nums[i]<nums[i-1]) swap(nums[i],nums[i-1]);

（2）如果i为谷的位置，则判断当前位置是否大于前一个位置（前一个为峰），若大于，则交换，大于则不处理。即： if(nums[i]>nums[i-1]) swap(nums[i],nums[i-1]);

class Solution {

public:

    void wiggleSort(vector<int>& nums) {

        for(int i=1;i<nums.size();i++){

            if(i%2==0){

                if(nums[i]<nums[i-1]) swap(nums[i],nums[i-1]);

            }

            else{

                if(nums[i]>nums[i-1]) swap(nums[i],nums[i-1]);

            }

        }

    }

};

[面试题 16.01. 交换数字](https://leetcode-cn.com/problems/swap-numbers-lcci/)

编写一个函数，不用临时变量，直接交换numbers = [a, b]中a与b的值。

**示例：输入:** numbers = [1,2]

**输出:** [2,1]

class Solution {

public:

    vector<int> swapNumbers(vector<int>& numbers) {

        numbers[0] ^= numbers[1];

        numbers[1] ^= numbers[0];

        numbers[0] ^= numbers[1];

        return numbers;

    }

};

[面试题 16.02. 单词频率](https://leetcode-cn.com/problems/words-frequency-lcci/)

设计一个方法，找出任意指定单词在一本书中的出现频率。

你的实现应该支持如下操作：

* WordsFrequency(book)构造函数，参数为字符串数组构成的一本书
* get(word)查询指定单词在书中出现的频率

**示例：**

WordsFrequency wordsFrequency = new WordsFrequency({"i", "have", "an", "apple", "he", "have", "a", "pen"});

wordsFrequency.get("you"); //返回0，"you"没有出现过

wordsFrequency.get("have"); //返回2，"have"出现2次

wordsFrequency.get("an"); //返回1

wordsFrequency.get("apple"); //返回1

wordsFrequency.get("pen"); //返回1

class WordsFrequency {

struct TreeNode{

    int f;

    TreeNode\* n[26];

    TreeNode(): f(0){

        for(int i = 0; i < 26; ++i){

            n[i] = NULL;

        }

    }

};

    TreeNode\* root;

public:

    WordsFrequency(vector<string>& book) {

        root = new TreeNode();

        for(auto& x: book){

            auto temp = root;

            for(auto& c: x){

                if(!temp->n[c-'a']){

                    temp->n[c-'a'] = new TreeNode();

                }

                temp = temp->n[c-'a'];

            }

            ++(temp->f);

        }

    }

    int get(string word) {

        auto temp = root;

        for(auto& c: word){

            if(temp->n[c-'a'])

                temp = temp->n[c-'a'];

            else

                return 0;

        }

        return temp->f;

    }

};

[面试题 16.03. 交点](https://leetcode-cn.com/problems/intersection-lcci/)

给定两条线段（表示为起点start = {X1, Y1}和终点end = {X2, Y2}），如果它们有交点，请计算其交点，没有交点则返回空值。

要求浮点型误差不超过10^-6。若有多个交点（线段重叠）则返回 X 值最小的点，X 坐标相同则返回 Y 值最小的点。

**示例 1：**

**输入：**

line1 = {0, 0}, {1, 0}

line2 = {1, 1}, {0, -1}

**输出：** {0.5, 0}

**示例 2：**

**输入：**

line1 = {0, 0}, {3, 3}

line2 = {1, 1}, {2, 2}

**输出：** {1, 1}

**示例 3：**

**输入：**

line1 = {0, 0}, {1, 1}

line2 = {1, 0}, {2, 1}

**输出：** {}，两条线段没有交点

class Solution {

public:

    // 判断 (xk, yk) 是否在「线段」(x1, y1)~(x2, y2) 上

    // 这里的前提是 (xk, yk) 一定在「直线」(x1, y1)~(x2, y2) 上

    bool inside(int x1, int y1, int x2, int y2, int xk, int yk) {

        // 若与 x 轴平行，只需要判断 x 的部分

        // 若与 y 轴平行，只需要判断 y 的部分

        // 若为普通线段，则都要判断

        return (x1 == x2 || (min(x1, x2) <= xk && xk <= max(x1, x2))) && (y1 == y2 || (min(y1, y2) <= yk && yk <= max(y1, y2)));

    }

    void update(vector<double>& ans, double xk, double yk) {

        // 将一个交点与当前 ans 中的结果进行比较

        // 若更优则替换

        if (!ans.size() || xk < ans[0] || (xk == ans[0] && yk < ans[1])) {

            ans = {xk, yk};

        }

    }

    vector<double> intersection(vector<int>& start1, vector<int>& end1, vector<int>& start2, vector<int>& end2) {

        int x1 = start1[0], y1 = start1[1];

        int x2 = end1[0], y2 = end1[1];

        int x3 = start2[0], y3 = start2[1];

        int x4 = end2[0], y4 = end2[1];

        vector<double> ans;

        // 判断 (x1, y1)~(x2, y2) 和 (x3, y3)~(x4, y3) 是否平行

        if ((y4 - y3) \* (x2 - x1) == (y2 - y1) \* (x4 - x3)) {

            // 若平行，则判断 (x3, y3) 是否在「直线」(x1, y1)~(x2, y2) 上

            if ((y2 - y1) \* (x3 - x1) == (y3 - y1) \* (x2 - x1)) {

                // 判断 (x3, y3) 是否在「线段」(x1, y1)~(x2, y2) 上

                if (inside(x1, y1, x2, y2, x3, y3)) {

                    update(ans, (double)x3, (double)y3);

                }

                // 判断 (x4, y4) 是否在「线段」(x1, y1)~(x2, y2) 上

                if (inside(x1, y1, x2, y2, x4, y4)) {

                    update(ans, (double)x4, (double)y4);

                }

                // 判断 (x1, y1) 是否在「线段」(x3, y3)~(x4, y4) 上

                if (inside(x3, y3, x4, y4, x1, y1)) {

                    update(ans, (double)x1, (double)y1);

                }

                // 判断 (x2, y2) 是否在「线段」(x3, y3)~(x4, y4) 上

                if (inside(x3, y3, x4, y4, x2, y2)) {

                    update(ans, (double)x2, (double)y2);

                }

            }

            // 在平行时，其余的所有情况都不会有交点

        } else {

            // 联立方程得到 t1 和 t2 的值

            double t1 = (double)(x3 \* (y4 - y3) + y1 \* (x4 - x3) - y3 \* (x4 - x3) - x1 \* (y4 - y3)) / ((x2 - x1) \* (y4 - y3) - (x4 - x3) \* (y2 - y1));

            double t2 = (double)(x1 \* (y2 - y1) + y3 \* (x2 - x1) - y1 \* (x2 - x1) - x3 \* (y2 - y1)) / ((x4 - x3) \* (y2 - y1) - (x2 - x1) \* (y4 - y3));

            // 判断 t1 和 t2 是否均在 [0, 1] 之间

            if (t1 >= 0.0 && t1 <= 1.0 && t2 >= 0.0 && t2 <= 1.0) {

                ans = {x1 + t1 \* (x2 - x1), y1 + t1 \* (y2 - y1)};

            }

        }

        return ans;

    }

};

[面试题 16.04. 井字游戏](https://leetcode-cn.com/problems/tic-tac-toe-lcci/)

设计一个算法，判断玩家是否赢了井字游戏。输入是一个 N x N 的数组棋盘，由字符" "，"X"和"O"组成，其中字符" "代表一个空位。以下是井字游戏的规则：

* 玩家轮流将字符放入空位（" "）中。
* 第一个玩家总是放字符"O"，且第二个玩家总是放字符"X"。
* "X"和"O"只允许放置在空位中，不允许对已放有字符的位置进行填充。
* 当有N个相同（且非空）的字符填充任何行、列或对角线时，游戏结束，对应该字符的玩家获胜。
* 当所有位置非空时，也算为游戏结束。如果游戏结束，玩家不允许再放置字符。

如果游戏存在获胜者，就返回该游戏的获胜者使用的字符（"X"或"O"）；如果游戏以平局结束，则返回 "Draw"；如果仍会有行动（游戏未结束），则返回 "Pending"。

**示例 1：**

**输入：** board = ["O X"," XO","X O"]

**输出：** "X"

**示例 2：**

**输入：** board = ["OOX","XXO","OXO"]

**输出：** "Draw"

**解释：** 没有玩家获胜且不存在空位

**示例 3：**

**输入：** board = ["OOX","XXO","OX "]

**输出：** "Pending"

**解释：** 没有玩家获胜且仍存在空位

class Solution {

public:

    string tictactoe(vector<string>& board) {

        int bSize=board.size(), N = board.size();

        int sum\_row=0, sum\_col=0,sum\_dia\_right=0,sum\_dia\_left =0,isFull=1;

        for(int i=0;i<bSize;i++){

            sum\_row=0, sum\_col=0;

            sum\_dia\_left += board[i][i];           //主对角相加

            sum\_dia\_right += board[i][bSize-1-i];  //副对角相加

            for(int j=0; j<bSize; j++){

                sum\_row += board[i][j]; //行相加

                sum\_col += board[j][i];;//列相加

                if(board[i][j] == ' ') isFull =0;

            }

            if(sum\_row == ((int)'X') \* N || sum\_col == ((int)'X') \* N){

                return string("X");

            }

            if(sum\_row == ((int)'O') \* N || sum\_col == ((int)'O') \* N){

                return string("O");

            }

        }

        if(sum\_dia\_left == ((int)'X') \* N || sum\_dia\_right == ((int)'X') \* N){

                return string("X");

        }

        if(sum\_dia\_left == ((int)'O') \* N || sum\_dia\_right == ((int)'O') \* N){

                return string("O");

        }

        if(isFull){ //没有选手获胜

            return string("Draw");

        }else{

            return string("Pending");

        }

    }

};

[面试题 16.05. 阶乘尾数](https://leetcode-cn.com/problems/factorial-zeros-lcci/)

设计一个算法，算出 n 阶乘有多少个尾随零。

**示例 1:**

**输入:** 3

**输出:** 0

**解释:** 3! = 6, 尾数中没有零。

**示例 2:**

**输入:** 5

**输出:** 1

**解释:** 5! = 120, 尾数中有 1 个零.

1、最初想到先把阶乘计算出来，再对结果进行处理，从而得到0的个数，但当计算完阶乘后，发现0的个数并不好获取。

2、之后考虑到，哪些数相乘会有零出现，也就是哪些数相乘会是10或10的倍数。也就得到2x5、4x5、6x5......会是10或10的倍数,也就是5和一个偶数相乘会得到10或10的倍数。

3、在数的阶乘中，偶数的个数明显多于5的个数，所以只要计算出阶乘中每个数的因数中5的个数，即可得到0的个数

4、所以只要计算在阶乘中是5的倍数的个数（包含一个5）、25的倍数的个数（包含两个5）、125的倍数的个数（包含3个5）......

class Solution {

public:

    int trailingZeroes(int n) {

        int sum = 0;

        while (n >= 5) {

            n /= 5;

            sum += n;

        }

        return sum;

    }

};

[面试题 16.06. 最小差](https://leetcode-cn.com/problems/smallest-difference-lcci/)

给定两个整数数组a和b，计算具有最小差绝对值的一对数值（每个数组中取一个值），并返回该对数值的差

**示例：输入：**{1, 3, 15, 11, 2}, {23, 127, 235, 19, 8}

**输出：** 3，即数值对(11, 8)

class Solution {

public:

    int smallestDifference(vector<int>& a, vector<int>& b)

    {

        sort(a.begin(),a.end());

        sort(b.begin(),b.end());

        long ret = INT\_MAX;

        for(int i = 0, j = 0; i < a.size() && j < b.size();)

        {

            ret = min(ret,abs(long(a[i])-long(b[j])));

            if(a[i] < b[j]){

                i++;

            }else{

                j++;

            }

        }

        return ret;

    }

};

[面试题 16.07. 最大数值](https://leetcode-cn.com/problems/maximum-lcci/)

编写一个方法，找出两个数字a和b中最大的那一个。不得使用if-else或其他比较运算符。

**示例：**

**输入：** a = 1, b = 2

**输出：** 2

本质是平均值法： max(a, b) = ((a + b) + abs(a - b)) / 2。

class Solution {

public:

    int maximum(int a, int b) {

        long c = a;

        long d = b;

        int res = (int) ((fabs(c-d) + c + d)/2);

        return res;

    }

};

绝对值的位运算

为了回避abs，利用位运算实现绝对值功能。

以int8\_t为例：分析运算：(var ^ (var >> 7)) - (var >> 7)

var >= 0: var >> 7 => 0x00，即：(var ^ 0x00) - 0x00，异或结果为var

var < 0: var >> 7 => 0xFF，即：(var ^ 0xFF) - 0xFF，var ^ 0xFF是在对var的全部位取反，-0xFF <=> +1, 对signed int取反加一就是取其相反数。

举个栗子：var = -3 <=> 0xFD，(var ^ 0xFF) - 0xFF= 0x02 - 0xff= 0x03

类型      绝对值位运算

int8\_t      (var ^ (var >> 7)) - (var >> 7)

int16\_t     (var ^ (var >> 15)) - (var >> 15)

int32\_t     (var ^ (var >> 31)) - (var >> 31)

int64\_t     (var ^ (var >> 63)) - (var >> 63)

代码中(\_diff ^ (\_diff >> 63)) - (\_diff >> 63)就是在求取long (int64\_t)的绝对值。

class Solution {

public:

    int maximum(int a, int b) {

        long \_sum = long(a) + long(b);

        long \_diff = long(a) - long(b);

        long \_abs\_diff = (\_diff ^ (\_diff >> 63)) - (\_diff >> 63);

        return (\_sum + \_abs\_diff) / 2;

    }

};

[面试题 16.08. 整数的英语表示](https://leetcode-cn.com/problems/english-int-lcci/)

给定一个整数，打印该整数的英文描述。

**示例 1:**

**输入:** 123

**输出:** "One Hundred Twenty Three"

**示例 2:**

**输入:** 12345

**输出:** "Twelve Thousand Three Hundred Forty Five"

**示例 3:**

**输入:** 1234567

**输出:** "One Million Two Hundred Thirty Four Thousand Five Hundred Sixty Seven"

**示例 4:**

**输入:** 1234567891

**输出:** "One Billion Two Hundred Thirty Four Million Five Hundred Sixty Seven Thousand Eight Hundred Ninety One"

class Solution {

public:

    const int N[31] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 30, 40, 50, 60, 70, 80, 90,

            100, 1000, 1000000, 1000000000};

    const string S[31] = {"One", "Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten",

            "Eleven", "Twelve", "Thirteen", "Fourteen", "Fifteen", "Sixteen", "Seventeen", "Eighteen", "Nineteen",

            "Twenty", "Thirty", "Forty", "Fifty", "Sixty", "Seventy", "Eighty", "Ninety",

            "Hundred", "Thousand", "Million", "Billion"};

    const int K = 90;

    string numberToWords(int num) {

        if (num == 0) return "Zero";

        int i = 30;

        while (i >= 0 && N[i] > num) --i;

        string res;

        if (N[i] <= K) {

            res += S[i];

        } else {

            res += numberToWords(num / N[i]) + " " + S[i];

        }

        if (num % N[i] > 0) res += " " + numberToWords(num % N[i]);

        return res;

    }

};

[面试题 16.09. 运算](https://leetcode-cn.com/problems/operations-lcci/)

请实现整数数字的乘法、减法和除法运算，运算结果均为整数数字，程序中只允许使用加法运算符和逻辑运算符，允许程序中出现正负常数，不允许使用位运算。

你的实现应该支持如下操作：

* Operations() 构造函数
* minus(a, b) 减法，返回a - b
* multiply(a, b) 乘法，返回a \* b
* divide(a, b) 除法，返回a / b

**示例：**

Operations operations = new Operations();

operations.minus(1, 2); //返回-1

operations.multiply(3, 4); //返回12

operations.divide(5, -2); //返回-2

class Operations {

private:

    vector<int> negs, poss; // 存的是[-1, -2, -4...]、[1, 2, 4...]，取反和判断溢出时用

    int neg(int a) {

        if(!a) return 0;

        int result = 0;

        if(a > 0) {

            // 从绝对值最大的部分开始填充

            for(auto p = negs.rbegin(); p != negs.rend(); p++) {

                if(\*p + a < 0) continue;

                a += \*p;

                result += \*p;

            }

        } else {

            for(auto p = poss.rbegin(); p != poss.rend(); p++) {

                if(\*p + a > 0) continue;

                a += \*p;

                result += \*p;

            }

        }

        return result;

    }

public:

    Operations() {

        // 构造poss和negs

        int p = 1, n = -1;

        poss.push\_back(p);

        negs.push\_back(n);

        for(int i = 0; i < 30; i++) {

            p += p;

            n += n;

            poss.push\_back(p);

            negs.push\_back(n);

        }

    }

   int minus(int a, int b) {

        return a + neg(b);

}

int multiply(int a, int b) {

        if(!a || !b) return 0;

        if(a == 1) return b;        // 这一步是针对b = INT\_MIN的情况，防止下一步取neg时溢出

        if(b < 0) return neg(multiply(a, neg(b)));

        int result = a;

        int times = 1;              // times表示当前结果里已经累加了几个a了

        // times < poss[30]是为了防止溢出

        while(times < poss[30] && times + times <= b) {

            result += result;

            times += times;

        }

        result += multiply(a, minus(b, times));

        return result;

    }

    int divide(int a, int b) {

        if(!a) return 0;

        int result = 1;

        // 只写同号的情况，非同号时用neg转化成同号，但是要注意溢出

        if(a > 0) {

            if(b == INT\_MIN) return 0;          // 防止下一句取neg的时候溢出

            if(b < 0) return neg(divide(a, neg(b)));

            if(a < b) return 0;

            int acc = b;                        // 不断往acc里填充b，直到acc达到a

            while(acc < poss[30] && a >= acc + acc) {

                result += result;               // result表示已经填充了几个b了

                acc += acc;

            }

            result += divide(minus(a, acc), b);

        } else {

            if(b == 1) return a;                // 防止若a=INT\_MIN造成下一句运算时溢出

            if(b > 0) return neg(divide(a, neg(b)));

            if(a > b) return 0;

            int acc = b;

            while(acc >= negs[30] && a <= acc + acc) {

                result += result;

                acc += acc;

            }

            result += divide(minus(a, acc), b);

        }

        return result;

    }

};

[面试题 16.10. 生存人数](https://leetcode-cn.com/problems/living-people-lcci/)

给定N个人的出生年份和死亡年份，第i个人的出生年份为birth[i]，死亡年份为death[i]，实现一个方法以计算生存人数最多的年份。

你可以假设所有人都出生于1900年至2000年（含1900和2000）之间。如果一个人在某一年的任意时期都处于生存状态，那么他们应该被纳入那一年的统计中。例如，生于1908年、死于1909年的人应当被列入1908年和1909年的计数。

如果有多个年份生存人数相同且均为最大值，输出其中最小的年份。

**示例：**

**输入：**

birth = {1900, 1901, 1950}

death = {1948, 1951, 2000}

**输出：** 1901

很多这种类型的题，都是套路了，一个模式。

即使用一个前缀和(积)来保存某i之前所有的状态累计，如此可以求任意两个状态之间的累计。

i         ...0  ...  1  ...  50

生\_初始     1       1        1

生\_累计     1       2        3

i        ...48 ...  51  ...  100

死\_初始     1       1        1

死\_累计     1       2        3

class Solution {

public:

    int maxAliveYear(vector<int>& birth, vector<int>& death) {

        vector<int> lives(1001, 0), gos(1001,0);

        for(const int &val: birth) lives[val-1900]++;

        for(const int &val: death) gos[val-1900]++;

        for(int i = 1; i < 1001; i++) lives[i] = lives[i-1] + lives[i];

        for(int i = 1; i < 1001; i++) gos[i] = gos[i-1] + gos[i];

        int max\_p = lives[0];

        int year = 0;

        for(int i = 1; i < 1001; i++){

            int p = lives[i] - gos[i-1];

            if(max\_p < p){

                max\_p = p;

                year = i;

            }

        }

        return 1900+year;

    }

};

[面试题 16.11. 跳水板](https://leetcode-cn.com/problems/diving-board-lcci/)

你正在使用一堆木板建造跳水板。有两种类型的木板，其中长度较短的木板长度为shorter，长度较长的木板长度为longer。你必须正好使用k块木板。编写一个方法，生成跳水板所有可能的长度。

返回的长度需要从小到大排列。

**示例 1**

**输入：**

shorter = 1

longer = 2

k = 3

**输出：** [3,4,5,6]

**解释：**

可以使用 3 次 shorter，得到结果 3；使用 2 次 shorter 和 1 次 longer，得到结果 4 。以此类推，得到最终结果。

class Solution {

public:

    vector<int> divingBoard(int shorter, int longer, int k) {

        if (k == 0) {

            return vector <int> ();

        }

        if (shorter == longer) {

            return vector <int> (1, shorter \* k);

        }

        vector <int> lengths(k + 1);

        for (int i = 0; i <= k; ++i) {

            lengths[i] = shorter \* (k - i) + longer \* i;

        }

        return lengths;

    }

};

[面试题 16.13. 平分正方形](https://leetcode-cn.com/problems/bisect-squares-lcci/)

给定两个正方形及一个二维平面。请找出将这两个正方形分割成两半的一条直线。假设正方形顶边和底边与 x 轴平行。

每个正方形的数据square包含3个数值，正方形的左下顶点坐标[X,Y] = [square[0],square[1]]，以及正方形的边长square[2]。所求直线穿过两个正方形会形成4个交点，请返回4个交点形成线段的两端点坐标（两个端点即为4个交点中距离最远的2个点，这2个点所连成的线段一定会穿过另外2个交点）。2个端点坐标[X1,Y1]和[X2,Y2]的返回格式为{X1,Y1,X2,Y2}，要求若X1 != X2，需保证X1 < X2，否则需保证Y1 <= Y2。

若同时有多条直线满足要求，则选择斜率最大的一条计算并返回（与Y轴平行的直线视为斜率无穷大）。

**示例：**

**输入：**

square1 = {-1, -1, 2}

square2 = {0, -1, 2}

**输出：** {-1,0,2,0}

**解释：** 直线 y = 0 能将两个正方形同时分为等面积的两部分，返回的两线段端点为[-1,0]和[2,0]

class Solution {

public:

    vector<double> cutSquares(vector<int>& square1, vector<int>& square2) {

        // 计算两个正方形中心(cx1, cy1), (cx2, cy2)

        double cx1 = square1[0] + ((double)square1[2]) / 2;

        double cy1 = square1[1] + ((double)square1[2]) / 2;

        double cx2 = square2[0] + ((double)square2[2]) / 2;

        double cy2 = square2[1] + ((double)square2[2]) / 2;

        vector<pair<double, double> > res; // 候选顶点集

        // 处理特殊情况cx1 == cx2和cy1 == cy2

        if (cx1 == cx2) {

            res.push\_back({cx1, square1[1]});

            res.push\_back({cx1, square2[1]});

            res.push\_back({cx1, square1[1]+square1[2]});

            res.push\_back({cx1, square2[1]+square2[2]});

        } else if (cy1 == cy2) {

            res.push\_back({square1[0], cy1});

            res.push\_back({square2[0], cy1});

            res.push\_back({square1[0]+square1[2], cy1});

            res.push\_back({square2[0]+square2[2], cy1});

        } else {

            // 直线方程f(y)

            auto fy = [=](double y) -> double {

                const double k = (cx2 - cx1) / (cy2 - cy1);

                return k \* (y - cy1) + cx1;

            };

            // 直线方程f(x)

            auto fx = [=](double x) -> double {

                const double k = (cy2 - cy1) / (cx2 - cx1);

                return k \* (x - cx1) + cy1;

            };

            for (auto &sq : {square1, square2}) {

                for (auto &p : vector<pair<double, double> >{

                    {fy(sq[1]), sq[1]},

                    {fy(sq[1]+sq[2]), sq[1]+sq[2]},

                    {sq[0], fx(sq[0])},

                    {sq[0]+sq[2], fx(sq[0]+sq[2])}})

                {

                    // 判断候选顶点是否在正方形中

                    if (p.first >= sq[0] && p.first <= sq[0] + sq[2]) {

                        if (p.second >= sq[1] && p.second <= sq[1] + sq[2]) {

                            res.push\_back(p);

                        }

                    }

                }

            }

        }

        // 对顶点集排序

        sort(res.begin(), res.end());

        // 所求顶点为排序后的`res`数组的第一个顶点和最后一个顶点。

        return {

            res.front().first,

            res.front().second,

            res.back().first,

            res.back().second,

        };

    }

};

[面试题 16.14. 最佳直线](https://leetcode-cn.com/problems/best-line-lcci/)

给定一个二维平面及平面上的 N 个点列表Points，其中第i个点的坐标为Points[i]=[Xi,Yi]。请找出一条直线，其通过的点的数目最多。

设穿过最多点的直线所穿过的全部点编号从小到大排序的列表为S，你仅需返回[S[0],S[1]]作为答案，若有多条直线穿过了相同数量的点，则选择S[0]值较小的直线返回，S[0]相同则选择S[1]值较小的直线返回。

**示例：**

**输入：** [[0,0],[1,1],[1,0],[2,0]]

**输出：** [0,2]

**解释：** 所求直线穿过的3个点的编号为[0,2,3]

// 记录直线上的点的个数，以及最小的两个编号

struct Record {

  vector<int> ids;

  int cnt;

  Record() : cnt(0) {

  }

};

using LL = long long;

LL gcd(LL a, LL b) {

  LL r;

  do {

    r = a % b;

    a = b;

    b = r;

  } while (r != 0);

  return a;

}

class Solution {

public:

  vector<int> bestLine(vector<vector<int>>& points) {

    // (x - x1) / (x2 - x1) = (y - y1) / (y2 - y1)

    // (y2 - y1) x + (x1 - x2) y = (y2 - y1) \* x1 + (x1 - x2) \* y1

    int n = points.size();

    map<array<LL, 3>, Record> ABC\_ma;

    int best = 0;

    vector<int> ans;

    // 如果A与B和A与C都在一条直线上，那么B与C也在该条直线上

    for (int i = 0; i < n; ++i) {

      vector<int> &pi = points[i];

      int dup = 0; // 与pi位置相同的点数

      ABC\_ma.clear();

      int local\_best = 0;

      vector<int> local\_ans;

      for (int j = i + 1; j < n; ++j) {

        vector<int> &pj = points[j];

        LL A = (LL)pj[1] - pi[1];

        LL B = (LL)pj[0] - pi[0];

        if (A == 0 && B == 0) {

          // pi与pj相同

          ++dup;

          if (local\_ans.empty()) local\_ans = {i, j};

          continue;

        }

        LL C = A \* pi[0] + B \* pi[1];

        if (A == 0) {

          if (C == 0) {

            B = 1;

          } else {

            LL g = gcd(B, C);

            B /= g;

            C /= g;

          }

        } else if (B == 0) {

          if (C == 0) {

            A = 1;

          } else {

            LL g = gcd(A, C);

            A /= g;

            C /= g;

          }

        } else {

          // A != 0 and B != 0

          LL g = gcd(A, B);

          if (C == 0) {

            A /= g;

            B /= g;

          } else {

            LL w = gcd(g, C);

            A /= w;

            B /= w;

            C /= w;

          }

        }

        Record &r = ABC\_ma[{A, B, C}];

        if (r.cnt == 0) {

          r.ids = {i, j};

        }

        ++r.cnt;

      }

      update\_ans(ABC\_ma, local\_best, local\_ans);

      local\_best += dup;

      if (local\_best > best) {

        ans = std::move(local\_ans);

        best = local\_best;

      }

    }

    return ans;

  }

  template <typename T>

  void update\_ans(map<T, Record> &ma, int &best, vector<int> &ans) {

    for (auto &p : ma) {

      Record &r = p.second;

      if (r.cnt > best) {

        best = r.cnt;

        ans = std::move(r.ids);

      } else if (p.second.cnt == best) {

        // 如果穿过相同数量的点

        if (r.ids[0] < ans[0] || (r.ids[0] == ans[0] && r.ids[1] < ans[1])) {

          ans = std::move(r.ids);

        }

      }

    }

  }

};

[面试题 16.15. 珠玑妙算](https://leetcode-cn.com/problems/master-mind-lcci/)

珠玑妙算游戏（the game of master mind）的玩法如下。

计算机有4个槽，每个槽放一个球，颜色可能是红色（R）、黄色（Y）、绿色（G）或蓝色（B）。例如，计算机可能有RGGB 4种（槽1为红色，槽2、3为绿色，槽4为蓝色）。作为用户，你试图猜出颜色组合。打个比方，你可能会猜YRGB。要是猜对某个槽的颜色，则算一次“猜中”；要是只猜对颜色但槽位猜错了，则算一次“伪猜中”。注意，“猜中”不能算入“伪猜中”。

给定一种颜色组合solution和一个猜测guess，编写一个方法，返回猜中和伪猜中的次数answer，其中answer[0]为猜中的次数，answer[1]为伪猜中的次数。

**示例：**

**输入：** solution="RGBY",guess="GGRR"

**输出：** [1,1]

**解释：** 猜中1次，伪猜中1次。

class Solution {

public:

    vector<int> masterMind(string solution, string guess) {

        unordered\_map<char, int> chars;

        int ans1 = 0, total = 0;

        for(int i = 0; i < 4; i ++)

            chars[guess[i]] ++;

        for(int i = 0; i < 4; i ++){

            if(guess[i] == solution[i]){

                ans1 ++;

            }

            if(chars[solution[i]]) {

                total ++;

                chars[solution[i]] --;

            }

        }

        return {ans1, total - ans1};

    }

};

[面试题 16.16. 部分排序](https://leetcode-cn.com/problems/sub-sort-lcci/)

给定一个整数数组，编写一个函数，找出索引m和n，只要将索引区间[m,n]的元素排好序，整个数组就是有序的。注意：n-m尽量最小，也就是说，找出符合条件的最短序列。函数返回值为[m,n]，若不存在这样的m和n（例如整个数组是有序的），请返回[-1,-1]。

**示例：输入：** [1,2,4,7,10,11,7,12,6,7,16,18,19]

**输出：** [3,9]

首先虽然题目没说，但是实际运行下来数列是单调递增的，所以我们下面默认数列是递增的。

那么对于元素 a[i] 来说，如果它左边存在大于 a[i] 的元素，那么 a[i] 是一定要参与到排序里去的。或者说如果它右边存在小于 a[i] 的元素，那么 a[i] 也是要参与到排序里去的。

所以我们只需要寻找最靠右的那个数（满足左边存在大于它的数），和最靠左的那个数（满足右边存在小于它的数），那么这两个数之间就是要排序的区间了。

为什么最靠右的那个（满足左边存在大于它的数）数一定能保证右边没有更小的数了呢？因为如果右边还有更小的数，那么那个更小的数才是更靠右的啊，这就矛盾了。

所以我们只需要从左到右扫描一遍，用一个变量维护一下最大值就行了，然后反向再遍历一遍，维护一个最小值。

class Solution {

public:

    vector<int> subSort(vector<int>& array) {

        int n = array.size();

        int maxx = INT\_MIN, minn = INT\_MAX;

        int l = -1, r = -1;

        for (int i = 0; i < n; ++i) {

            if (array[i] < maxx) r = i;

            else maxx = array[i];

        }

        for (int i = n-1; i >= 0; --i) {

            if (array[i] > minn) l = i;

            else minn = array[i];

        }

        return {l, r};

    }

};

[面试题 16.17. 连续数列](https://leetcode-cn.com/problems/contiguous-sequence-lcci/)

给定一个整数数组，找出总和最大的连续数列，并返回总和。

**示例：**

**输入：** [-2,1,-3,4,-1,2,1,-5,4]

**输出：** 6

**解释：** 连续子数组 [4,-1,2,1] 的和最大，为 6。

class Solution {

public:

    int maxSubArray(vector<int>& nums) {

        int res = nums[0];

        for(int i = 1; i < nums.size(); ++i){

            nums[i] = max(nums[i - 1] + nums[i], nums[i]);

            res = max(nums[i], res);

        }

        return res;

    }

};

// 分治法

int maxSubArray(vector<int>& nums)

{

    if(nums.size() == 0) return INT\_MIN;

    return divide(nums,0,nums.size()-1);

}

int divide(vector<int>& nums, int left, int right)

{

    if(left == right) return nums[left];

    int mid = (left + right) / 2;

      int sumLeft = divide(nums,left,mid);   // 1. 最大数列和在左边

    int sumRight = divide(nums,mid+1,right);  // 2. 最大数列和在右边

    int leftSum = 0,leftMaxSum = INT\_MIN;     // 3. 最大数列和在中间。先求左边的最大和

    for(int i = mid; i >= left; i--)

    {

        leftSum += nums[i];

        leftMaxSum = max(leftMaxSum,leftSum);

    }

    int rightSum = 0,rightMaxSum = INT\_MIN;     // 求右边的最大和

    for(int i = mid + 1; i <= right; i++)

    {

        rightSum += nums[i];

        rightMaxSum = max(rightMaxSum,rightSum);

    }

    return max(max(sumLeft,sumRight),leftMaxSum+rightMaxSum);

}

[面试题 16.18. 模式匹配](https://leetcode-cn.com/problems/pattern-matching-lcci/)

你有两个字符串，即pattern和value。 pattern字符串由字母"a"和"b"组成，用于描述字符串中的模式。例如，字符串"catcatgocatgo"匹配模式"aabab"（其中"cat"是"a"，"go"是"b"），该字符串也匹配像"a"、"ab"和"b"这样的模式。但需注意"a"和"b"不能同时表示相同的字符串。编写一个方法判断value字符串是否匹配pattern字符串。

**示例 1：**

**输入：** pattern = "abba", value = "dogcatcatdog"

**输出：** true

**示例 2：**

**输入：** pattern = "abba", value = "dogcatcatfish"

**输出：** false

**示例 3：**

**输入：** pattern = "aaaa", value = "dogcatcatdog"

**输出：** false

**示例 4：**

**输入：** pattern = "abba", value = "dogdogdogdog"

**输出：** true

**解释：** "a"="dogdog",b=""，反之也符合规则

class Solution {

public:

    bool patternMatching(string pattern, string value) {

        int count\_a = 0, count\_b = 0;

        for (char ch: pattern) {

            if (ch == 'a') {

                ++count\_a;

            } else {

                ++count\_b;

            }

        }

        if (count\_a < count\_b) {

            swap(count\_a, count\_b);

            for (char& ch: pattern) {

                ch = (ch == 'a' ? 'b' : 'a');

            }

        }

        if (value.empty()) {

            return count\_b == 0;

        }

        if (pattern.empty()) {

            return false;

        }

        for (int len\_a = 0; count\_a \* len\_a <= value.size(); ++len\_a) {

            int rest = value.size() - count\_a \* len\_a;

            if ((count\_b == 0 && rest == 0) || (count\_b != 0 && rest % count\_b == 0)) {

                int len\_b = (count\_b == 0 ? 0 : rest / count\_b);

                int pos = 0;

                bool correct = true;

                string value\_a, value\_b;

                for (char ch: pattern) {

                    if (ch == 'a') {

                        string sub = value.substr(pos, len\_a);

                        if (!value\_a.size()) {

                            value\_a = move(sub);

                        } else if (value\_a != sub) {

                            correct = false;

                            break;

                        }

                        pos += len\_a;

                    } else {

                        string sub = value.substr(pos, len\_b);

                        if (!value\_b.size()) {

                            value\_b = move(sub);

                        } else if (value\_b != sub) {

                            correct = false;

                            break;

                        }

                        pos += len\_b;

                    }

                }

                if (correct && value\_a != value\_b) {

                    return true;

                }

            }

        }

        return false;

    }

};

[面试题 16.19. 水域大小](https://leetcode-cn.com/problems/pond-sizes-lcci/)

你有一个用于表示一片土地的整数矩阵land，该矩阵中每个点的值代表对应地点的海拔高度。若值为0则表示水域。由垂直、水平或对角连接的水域为池塘。池塘的大小是指相连接的水域的个数。编写一个方法来计算矩阵中所有池塘的大小，返回值需要从小到大排序。

**示例：**

**输入：**

[

[0,2,1,0],

[0,1,0,1],

[1,1,0,1],

[0,1,0,1]

]

**输出：** [1,2,4]

class Solution {

public:

    vector<int> pondSizes(vector<vector<int>>& land) {

        vector<int> res;

        for(int i = 0;i < land.size();i++)

        {

            for(int j = 0;j < land[i].size();j++)

            {

                int count = 0;

                if(land[i][j] == 0)

                {

                    dfs(i,j,land,count);//有效的一次dfs

                    if(count != 0)//计算一片鱼塘大小

                    res.push\_back(count);

                }

            }

        }

        sort(res.begin(),res.end());

        return res;

    }

    void dfs(int x,int y,vector<vector<int>> &land,int &count)

    {

        land[x][y] = -1;//染色

        count++;

        int dx[8] = {0,1,1,1,0,-1,-1,-1},dy[8] = {1,1,0,-1,-1,-1,0,1};//方向数组

        for(int i = 0;i < 8;i++)

        {

            int x1 = dx[i] + x;

            int y1 = dy[i] + y;

            if(x1 >= 0 && x1 < land.size() && y1 >= 0 && y1 <land[0].size() && land[x1][y1] == 0)//在方格内且是鱼塘

            dfs(x1,y1,land,count);//再次搜索

        }

    }

};

[面试题 16.20. T9键盘](https://leetcode-cn.com/problems/t9-lcci/)

在老式手机上，用户通过数字键盘输入，手机将提供与这些数字相匹配的单词列表。每个数字映射到0至4个字母。给定一个数字序列，实现一个算法来返回匹配单词的列表。你会得到一张含有有效单词的列表。映射如下图所示：



**示例 1:**

**输入:** num = "8733", words = ["tree", "used"]

**输出:** ["tree", "used"]

**示例 2:**

**输入:** num = "2", words = ["a", "b", "c", "d"]

**输出:** ["a", "b", "c"]

class Solution {

public:

    vector<string> getValidT9Words(string num, vector<string>& words) {

        vector<string> ans;

        int n=num.size(),w=words.size();

        if(n==0||w==0) return ans;

        vector<char> table={'2','2','2','3','3','3','4','4','4','5','5','5','6','6','6','7','7','7','7','8',  '8','8','9','9','9','9'};//'t'-'a'

        for(string s:words){

            if(s.size()==n){

                int i=0;

                for(;i<n;i++){

                    if(table[int(s[i]-'a')]!=num[i]) break;

                }

                if(i==n) ans.push\_back(s);

            }

        }

        return ans;

    }

};

[面试题 16.21. 交换和](https://leetcode-cn.com/problems/sum-swap-lcci/)

给定两个整数数组，请交换一对数值（每个数组中取一个数值），使得两个数组所有元素的和相等。

返回一个数组，第一个元素是第一个数组中要交换的元素，第二个元素是第二个数组中要交换的元素。若有多个答案，返回任意一个均可。若无满足条件的数值，返回空数组。

**示例:**

**输入:** array1 = [4, 1, 2, 1, 1, 2], array2 = [3, 6, 3, 3]

**输出:** [1, 3]

**示例:**

**输入:** array1 = [1, 2, 3], array2 = [4, 5, 6]

**输出:** []

sum\_a - x = sum\_b + x

sum\_a - sum\_b = 2 \* x

即: 两数组和 之差必须为偶数,才可能交换, 否则返回 [ ]

确定数组和差 为 (diff = sum\_a - sum\_b) // 2,

遍历其中一个,在另一个中查找即可. 转为 两数之和,一样的解法.

class Solution {

public:

   vector<int> findSwapValues(vector<int>& array1, vector<int>& array2) {

        set<int>s;

        int sum1=0,sum2=0;

        for(int i=0;i<array1.size();i++)

            sum1+=array1[i];

        for(int i=0;i<array2.size();i++){

            sum2+=array2[i];

            s.insert(array2[i]);

        }

        if(abs(sum2-sum1)%2==1)     //abs取绝对值

           return {};                      //奇数直接返回空数组

        int n=(sum2-sum1)/2;                //体会一下这里n如果是负数会怎样

        for(int i=0;i<array1.size();i++){

            if(s.find(array1[i]+n)!=s.end())      //找到了就返回这俩数就行了

                return {array1[i],array1[i]+n}; //这里用array[i]+n代替array2里需要交换的数，秒！

        }

        return {};      //没找到返回空数组

    }

};

[面试题 16.22. 兰顿蚂蚁](https://leetcode-cn.com/problems/langtons-ant-lcci/)

一只蚂蚁坐在由白色和黑色方格构成的无限网格上。开始时，网格全白，蚂蚁面向右侧。每行走一步，蚂蚁执行以下操作。

(1) 如果在白色方格上，则翻转方格的颜色，向右(顺时针)转 90 度，并向前移动一个单位。  
(2) 如果在黑色方格上，则翻转方格的颜色，向左(逆时针方向)转 90 度，并向前移动一个单位。

编写程序来模拟蚂蚁执行的前 K 个动作，并返回最终的网格。

网格由数组表示，每个元素是一个字符串，代表网格中的一行，黑色方格由 'X' 表示，白色方格由 '\_' 表示，蚂蚁所在的位置由 'L', 'U', 'R', 'D' 表示，分别表示蚂蚁 左、上、右、下 的朝向。只需要返回能够包含蚂蚁走过的所有方格的最小矩形。

**示例 1:**

**输入:** 0

**输出:** ["R"]

**示例 2:**

**输入:** 2

**输出:**

[

  "\_X",

  "LX"

]

使用set类型的black来存储所有走过的且变为黑色的地板位置

首先假设起点为数组(0,0)点，后面会根据此进行偏移

black是动态变化的，因为白的能变成黑的，黑的也能变成白的

使用top\_left/top\_right/top\_up/top\_down分别记录当前走过的能达到的四个边界

由此可以得到最终结果数组的长宽大小

根据top\_left和top\_up可以获得数组零点的偏移，即对于black中的(x,y),其对应最终答案中的位置为(top\_up-y, x-top\_left);

为便于方向的转换，用vector按照顺时针方向记录一个数组{L,U,R,D}

此时，若当前方向为i，则右转下一个方向为(i+1)%4, 左转下一个方向为(i-1+4)%4

为便于下一个位置(x,y)的计算，与该方向数组的下标对应，设置hori/vert分别表示四个方向行走的时候(x,y)的变化情况

初始时，实际实在(1,0)位置，这一点需要注意

运行程序k次，不断更新black。这部分其实是最简单的。

最后计算结果数组的大小, 以及零点的偏移，把black中对应的板砖变黑

组后把当前位置更新上去即可

class Solution {

public:

    vector<string> printKMoves(int K) {

        set<pair<int,int>> black;

        if(K == 0) return vector<string>{"R"};

        vector<char> direction = {'L', 'U', 'R', 'D'};

        vector<int> hori = {-1, 0, 1, 0};

        vector<int> vert = {0, 1, 0, -1};

        int cur\_x = 1, cur\_y = 0;

        int cur\_direction = 2;

        int top\_left = 1, top\_right = 0

        int top\_up = 0, top\_down = 0;

        for(int i = 0; i < K; i++){

            if(black.find(pair<int,int>{cur\_x, cur\_y}) == black.end()){

                black.insert(pair<int,int>{cur\_x, cur\_y});

                cur\_direction = (cur\_direction+1)%4;

            }else{

                black.erase(pair<int,int>{cur\_x, cur\_y});

                cur\_direction = (cur\_direction - 1 + 4) % 4;

            }

            cur\_x += hori[cur\_direction];

            cur\_y += vert[cur\_direction];

            top\_left = min(top\_left, cur\_x);

            top\_right = max(top\_right, cur\_x);

            top\_down = min(top\_down, cur\_y);

            top\_up = max(top\_up, cur\_y);

        }

        int rows = top\_up - top\_down + 1;

        int cols = top\_right - top\_left + 1;

        string t;

        for(int i = 0; i < cols; i++) t.push\_back('\_');

        vector<string> ans(rows, t);

        for(auto it=black.begin(); it != black.end(); it++){

            int x = it->first, y = it->second;

            ans[top\_up-y][x-top\_left] = 'X';

        }

        ans[top\_up-cur\_y][cur\_x-top\_left] = direction[cur\_direction];

        return ans;

    }

};

[面试题 16.24. 数对和](https://leetcode-cn.com/problems/pairs-with-sum-lcci/)

设计一个算法，找出数组中两数之和为指定值的所有整数对。一个数只能属于一个数对。

**示例 1:**

**输入:** nums = [5,6,5], target = 11

**输出:** [[5,6]]

**示例 2:**

**输入:** nums = [5,6,5,6], target = 11

**输出:** [[5,6],[5,6]]

//map

class Solution {

public:

    vector<vector<int>> pairSums(vector<int>& nums, int target)

    {

        vector<vector<int>> ans;

        unordered\_map<int, int> mp;

        for (auto n : nums)

            mp[n]++;

        for (auto n : nums)

            if (--mp[n] >= 0 && --mp[target - n] >= 0)

            {

                ans.push\_back({n, target - n});

                //mp[n]--, mp[target - n]--;

            }

        return ans;

    }

};

//双指针

vector<vector<int>> pairSums(vector<int>& nums, int target) {

    //取消同步特性，提高cin,cout效率

    vector<vector<int>> res;

    //排序

    sort(nums.begin(), nums.end());

    int i = 0, j = nums.size() - 1;

    while (i < j) {

        int sum = nums[i] + nums[j];

        if (sum == target) {

            res.push\_back({ nums[i],nums[j] });

            i++;

            j--;

        }

        else if (sum > target) j--;

        else i++;

    }

    return res;

}

[面试题 16.26. 计算器](https://leetcode-cn.com/problems/calculator-lcci/)

给定一个包含正整数、加(+)、减(-)、乘(\*)、除(/)的算数表达式(括号除外)，计算其结果。

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

**示例 1:**

**输入:** "3+2\*2"

**输出:** 7

**示例 2:**

**输入:** " 3/2 "

**输出:** 1

**示例 3:**

**输入:** " 3+5 / 2 "

**输出:** 5

class Solution {

public:

    int calculate(string s) {

        char op = '+';

        int val;

        istringstream iss(s);

        stack<int> st;

        while(iss>>val){

            if(op=='+'){

                st.push(val);

            }else if(op=='-'){

                st.push(-val);

            }else{

                int val2 = st.top(); st.pop();

                if(op=='\*') st.push(val\*val2);

                else if(op=='/') st.push(val2/val);

            }

            iss>>op;

        }

        int res = 0;

        while(st.size()){

            res += st.top(); st.pop();

        }

        return res;

    }

};

[面试题 17.01. 不用加号的加法](https://leetcode-cn.com/problems/add-without-plus-lcci/)

设计一个函数把两个数字相加。不得使用 + 或者其他算术运算符。

**示例:** **输入:** a = 1, b = 1 **输出:** 2

class Solution {

public:

    int add(int a, int b) {

        while(b){

            auto carry = (unsigned int)(a & b) << 1;

            a ^= b;

            b = carry;

        }

        return a;

    }

};

[面试题 17.04. 消失的数字](https://leetcode-cn.com/problems/missing-number-lcci/)

数组nums包含从0到n的所有整数，但其中缺了一个。请编写代码找出那个缺失的整数。你有办法在O(n)时间内完成吗？

**注意：**本题相对书上原题稍作改动

**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

class Solution {

public:

    int missingNumber(vector<int>& nums)

    {

        int sum = 0;

        for (int i = 0; i < nums.size(); i++)

        {

            sum ^= i;

            sum ^= nums[i];

        }

        sum ^= nums.size();

        return sum;

    }

};

[面试题 17.05. 字母与数字](https://leetcode-cn.com/problems/find-longest-subarray-lcci/)

给定一个放有字符和数字的数组，找到最长的子数组，且包含的字符和数字的个数相同。

返回该子数组，若存在多个最长子数组，返回左端点最小的。若不存在这样的数组，返回一个空数组。

**示例 1:**

**输入:** ["A","1","B","C","D","2","3","4","E","5","F","G","6","7","H","I","J","K","L","M"]

**输出:** ["A","1","B","C","D","2","3","4","E","5","F","G","6","7"]

**示例 2:**

**输入:** ["A","A"]

**输出:** []

数字看成-1，字母看成1，再计算前缀和。

前缀和 相同则计算下标的差值。

技巧：使用int[]数组memo存储 该前缀和 第1次出现时 的下标。

为何是第1次出现时的下标？

因为要求最长子数组。

比如：

["A","1","B","C","D","2","3","4","E","5","F","G","6","7","H","I","J","K","L","M"]

转化为[1, -1, 1, 1, 1, -1, -1, -1, 1, -1, 1, 1, -1, -1, 1, 1, 1, 1, 1, 1]

再转为前缀和形式[1, 0, 1, 2, 3, 2, 1, 0, 1, 0, 1, 2, 1, 0, 1, 2, 3, 4, 5, 6]。

前缀和相同是什么意思呢？

比如看加粗的2个1：[1, 0, 1, 2, 3, 2, 1, 0, 1, 0, 1, 2, 1, 0, 1, 2, 3, 4, 5, 6]

说明这2个1的下标所构成的区间内的数字和为0，而数字和为0说明有相同个数的1和-1，即相同个数的字母和数字。

具体来说：第1个加粗的1下标为0，第2个加粗的1下标为2，构成区间(0, 2](注意：是半开半闭)，

区间(0, 2]内元素的和即下标为1和下标为2的元素之和，[1, -1, 1, 1, 1, -1, -1, -1, 1, -1, 1, 1, -1, -1, 1, 1, 1, 1, 1, 1]

很明显-1 + 1 = 0

理解了这一点，剩余的工作就是找前缀和相同，且相隔最远的2个元素。

什么才是相隔最远？

[1, 0, 1, 2, 3, 2, 1, 0, 1, 0, 1, 2, 1, 0, 1, 2, 3, 4, 5, 6]

单只看前缀和为1的情况，毫无疑问，前缀和为1时，相隔最远的肯定是最左边的1和最右边的1.

因此我们只需要记录前缀和为1第一次出现时的下标即可(也就是记录了最左边的1的位置)，以后再遇到前缀和为1时，只需要将其下标 - 所记录的最左边的1的位置 即可。

至于memo数组大小为何初始化为(len << 1) + 1，考虑最极端的情况，要么全为-1，要么全为1，对应前缀和为-array.length和array.length。

特别注意:

当前缀和为0时，假设其下标为n，则说明区间[0, n]内所有元素的和为0，区间长度为n + 1。

因此将memo[0 + array.length]的值设为-1，因为n - (-1) = n + 1。

同时由于数组下标不能为负数，因此需要映射处理:

假设数组有2个元素，边界值为-2和2，即[-2, -1, 0, 1, 2]。

将[-2, -1, 0, 1, 2] 集体右移nums.length个单位，映射为[0, 1, 2, 3, 4]。

class Solution {

public:

    vector<string> findLongestSubarray(vector<string>& array) {

        int n=array.size();

        vector<int> prefix(n,0);

        unordered\_map<int,int> M;   //key,left\_index

        int left=0,right=-1;

        for(int i=0;i<n;++i){

            char ch=array[i][0];

            if(ch>='A' && ch<='z') prefix[i]=-1;

            else prefix[i]=1;

        }

        for(int i=1;i<n;++i){

            prefix[i]+=prefix[i-1];

        }

        for(int i=0;i<n;++i){

            auto it=M.find(prefix[i]);

            if(prefix[i]==0){

                if(right-left+1 < i+1){

                    right=i;left=0;

                }

                continue;

            }

            if(it==M.end()) M[prefix[i]]=i;

            else {

                if(right-left+1 < i-it->second){

                    right=i;left=it->second+1;

                }

            }

        }

        vector<string> ans;

        for(int i=left;i<=right;++i) ans.push\_back(array[i]);

        return ans;

    }

};

[面试题 17.06. 2出现的次数](https://leetcode-cn.com/problems/number-of-2s-in-range-lcci/)

编写一个方法，计算从 0 到 n (含 n) 中数字 2 出现的次数。

**示例:**

**输入:** 25

**输出:** 9

**解释:** (2, 12, 20, 21, 22, 23, 24, 25)(注意 22 应该算作两次)

解题思路

给定数字从个位开始向上分析

例如n=324时

- i     a      b      res

- 1     324    0      (324+7)/10\*1 + 0 = 33

- 10    32     4      (32+7)/10\*10 + 5 = 35

- 100   3      24     (3+7)/10\*100 = 100

res和为168即为所求

个位为4，5>2，故个位为2时，个位之前的取值有0~32共33种情况

十位为2，2为统计的数字，2为固定，百位取到0~2时各位才能取到0~9共310种情况，

而百位取3时，个位仅能取到0~4共五种情况，故为310+5 = 35种情况

百位为3，当百位取2时，后两位的取值有0~99共100种情况，故为100

同理，更大的数也是同理，“+7”是为特殊情况设计，当取值为0，1，2时不会产生进位，大于2时允许产生进位

class Solution {

public:

    int numberOf2sInRange(int n) {

        if(n <= 1)

            return 0;

        if(n <= 11)

            return 1;

        int a=0, b=0, res=0;

        for(long long i=1; i<=n; i\*=10){

            a = n/i;

            b = n%i;

            res += (a+7)/10\*i + (a%10 == 2)\*(b+1);

        }

        return res;

    }

};

[面试题 17.07. 婴儿名字](https://leetcode-cn.com/problems/baby-names-lcci/)

每年，政府都会公布一万个最常见的婴儿名字和它们出现的频率，也就是同名婴儿的数量。有些名字有多种拼法，例如，John 和 Jon 本质上是相同的名字，但被当成了两个名字公布出来。给定两个列表，一个是名字及对应的频率，另一个是本质相同的名字对。设计一个算法打印出每个真实名字的实际频率。注意，如果 John 和 Jon 是相同的，并且 Jon 和 Johnny 相同，则 John 与 Johnny 也相同，即它们有传递和对称性。

在结果列表中，选择**字典序最小**的名字作为真实名字。

**示例：**

**输入：**names = ["John(15)","Jon(12)","Chris(13)","Kris(4)","Christopher(19)"], synonyms = ["(Jon,John)","(John,Johnny)","(Chris,Kris)","(Chris,Christopher)"]

**输出：**["John(27)","Chris(36)"]

并查集，主要分为并操作和查操作

1、查操作，用来寻找祖先

2、并操作，能够将两个数据的祖先其中一个作为另一个的父节点，从而使得要合并的数据拥有同一个祖先

理解这两个操作就简单了，先初始化将每个数据的祖先设置为自己，接下来将要合并的数据进行并操作即可。最后只要找出祖先为自己的数据即可确定一共有哪些组。

class Solution {

public:

    unordered\_map<string, int> m\_name\_fre;//不需要排序应该用无序，比有序map效率提升不是一点半点

    unordered\_map<string, string> m\_parrent;

    string find(string name)//查操作，通过递归一直查到祖先，并且将路径上其他节点的父节点设置为祖先

    {

        if(m\_parrent.count(name) == 0)

            return name;

        string root = find(m\_parrent[name]);

        m\_parrent[name]=root;

        return root;

    }

    void m\_union(string n1, string n2)//并操作，将n1n2并为一类

    {

        n1 = find(n1);

        n2 = find(n2);

        if (n1 != n2)

        {

            //字典序比较

            if (n1 < n2)

                m\_parrent[n2] = n1;

            else

                m\_parrent[n1] = n2;

        }

    }

    vector<string> trulyMostPopular(const vector<string>& names, vector<string>& synonyms) {

        //对每个共同使用的名字对，进行并操作

        for (auto& name : synonyms)

        {

            int pos = name.find(',');

            string n1 = name.substr(1, pos - 1);

            string n2 = name.substr(pos + 1, name.size() - pos - 2);

            m\_union(n1,n2);

        }

        //将每个被使用的名字和其频率加入map

        for (auto& name : names)

        {

            int pos = name.find('(');

            string nm = name.substr(0, pos);

            int ifre = stoi(name.substr(pos + 1, name.size() - pos - 2));

            m\_name\_fre[find(nm)] += ifre;

        }

        //保存结果

        vector<string> result;

        for (auto& name : m\_name\_fre)

        {

            string fre = to\_string(name.second);

            result.push\_back(name.first + "(" + fre + ")");

        }

        return result;

    }

};

[面试题 17.08. 马戏团人塔](https://leetcode-cn.com/problems/circus-tower-lcci/)

有个马戏团正在设计叠罗汉的表演节目，一个人要站在另一人的肩膀上。出于实际和美观的考虑，在上面的人要比下面的人矮一点且轻一点。已知马戏团每个人的身高和体重，请编写代码计算叠罗汉最多能叠几个人。

**示例：**

**输入：**height = [65,70,56,75,60,68] weight = [100,150,90,190,95,110]

**输出：**6

**解释：**从上往下数，叠罗汉最多能叠 6 层：(56,90), (60,95), (65,100), (68,110), (70,150), (75,190)

题目给出两个维度，两个维度上都满足严格递增的要求才可以叠上去。

考虑需要严格递增，先按照height 升序排序，同时height 相同的人按照tweight 降序排序。

这样排序下来，直接在数组中查找关于weight 的最长递增子序列就能得到答案。

一个升序，一个降序的是与我们的解决方法相关（我们在排序之后，直接查找体重的最长递增子序列就能得到答案），

如： height = [3, 2, 2, 3, 1, 6]，weight = [7,3,5,6,2,10]。两个都升序的排序结果是height=[1,2,2,3,3,6]，weight=[2,3,5,6,7,10]，直接查找体重的最长递增子序列结果是6，

可以看到，同一身高内部可能存在体重的递增。因此，会被加入结果。而使用身高升序，体重降序，排序结果为height=[1,2,2,3,3,6]，weight=[2,5,3,7,6,10]，可以看到体重的最长递增子序列结果是4。

// 整个题目要求上面的人要比下面的人矮一点且轻一点，等价于下面的人要比上面的人矮一点且轻一点, 即反过来求

// 因为对于最优解路径, 正着求和反着求是同一个路径

// 为什么要反着求, 即从小到大, 那是因为对于std::lower\_bound()这个二分函数, 要求从小到大才可以二分

// 先固定第一维度(即高度按从小到大的顺序排序, 高度相同则体重从大到小), 然后再对体重这维dp(这个其实也可以看做贪心)

// dp[i]表示前i个数, 体重递增顺序的前提下, 第i个元素的体重尽量小

class Solution {

public:

    int bestSeqAtIndex(vector<int>& height, vector<int>& weight) {

        vector<pair<int,int>> tmp;

        for(int i = 0; i < height.size(); i++) tmp.push\_back({height[i], weight[i]});

        sort(tmp.begin(), tmp.end(), [](const pair<int,int> &a, const pair<int,int> &b) {

            return a.first == b.first ? a.second > b.second : a.first < b.first;

        });

        vector<int> dp; //长度为N的地方 最小的数字

        for(const auto &[h, w]: tmp) {

            auto p = lower\_bound(dp.begin(), dp.end(), w);  //二分查找第一个大于等于的地方

            if(p == dp.end()) dp.push\_back(w);

            else \*p = w;

        }

        return dp.size();

    }

};

[面试题 17.09. 第 k 个数](https://leetcode-cn.com/problems/get-kth-magic-number-lcci/)

有些数的素因子只有 3，5，7，请设计一个算法找出第 k 个数。注意，不是必须有这些素因子，而是必须不包含其他的素因子。例如，前几个数按顺序应该是 1，3，5，7，9，15，21。

**示例 1:** **输入:** k = 5**输出:** 9

class Solution {

public:

    int getKthMagicNumber(int k) {

        if (k <= 0) return 0;

        vector<long long int> nums(k+1, 1);  // 为防止越界，用 long long保存

        int p3 = 0, p5 = 0, p7 = 0;  // 标记"某个素数"的下标

        for (int i = 1; i < k; ++i)

        {

            nums[i] = min(min(3 \* nums[p3], 5 \* nums[p5]), 7 \* nums[p7]);

            if (nums[i] == 3 \* nums[p3]) p3++; // p3++是因为由p3所在的素数求得了最小值，故不会再由p3所在的素数求得另一个最小值，下一个最小值可能是3 \* nums[p3+1]。下面p5++, p7++同理。

            if (nums[i] == 5 \* nums[p5]) p5++;  // 注意此处是if,而不是else if,因为可能3 \*nums[p3] == 5 \* nums[p5] 或 7 \* nums[p7] == 5 \* nums[p5]。下面的同理。

            if (nums[i] == 7 \* nums[p7]) p7++;

        }

        return nums[k-1];

    }

};

[面试题 17.10. 主要元素](https://leetcode-cn.com/problems/find-majority-element-lcci/)

数组中占比超过一半的元素称之为主要元素。给定一个**整数**数组，找到它的主要元素。若没有，返回-1。

**示例 1：**

**输入：**[1,2,5,9,5,9,5,5,5]

**输出：**5

**示例 2：**

**输入：**[3,2]

**输出：**-1

**示例 3：**

**输入：**[2,2,1,1,1,2,2]

**输出：**2

class Solution {

public:

    int majorityElement(vector<int>& nums) {

        int count = 0, majority = 0;

        for(auto num : nums){

            if(!count)

                majority = num;

            if(num == majority)

                count++;

            else

                count--;

        }

        int C = 0;

        for(int i = 0; i < nums.size(); i++) {

            if(nums[i] == majority) C++;

        }

        if(C <= nums.size() / 2) majority = -1;

        return majority;

    }

};

[面试题 17.11. 单词距离](https://leetcode-cn.com/problems/find-closest-lcci/)

有个内含单词的超大文本文件，给定任意两个单词，找出在这个文件中这两个单词的最短距离(相隔单词数)。如果寻找过程在这个文件中会重复多次，而每次寻找的单词不同，你能对此优化吗?

**示例：**

**输入：**words = ["I","am","a","student","from","a","university","in","a","city"], word1 = "a", word2 = "student"

**输出：**1

class Solution {

public:

    int findClosest(vector<string>& words, string word1, string word2) {

        int t1 = -1, t2 = -1, res = words.size();

        for (int i = 0; i < words.size(); i ++) {

            if (words[i] == word1) t1 = i;

            else if (words[i] == word2) t2 = i;

            if (t1 != -1 && t2 != -1) res = min(res, abs(t1 - t2));

            if (res == 1) break;

        }

        return res;

    }

};

[面试题 17.12. BiNode](https://leetcode-cn.com/problems/binode-lcci/)

二叉树数据结构TreeNode可用来表示单向链表（其中left置空，right为下一个链表节点）。实现一个方法，把二叉搜索树转换为单向链表，要求依然符合二叉搜索树的性质，转换操作应是原址的，也就是在原始的二叉搜索树上直接修改。

返回转换后的单向链表的头节点。**注意：**本题相对原题稍作改动

**示例：输入：** [4,2,5,1,3,null,6,0]

**输出：** [0,null,1,null,2,null,3,null,4,null,5,null,6]

先定义一个节点ans用来返回整棵树，定义另一个节点cur=ans，用来表示正在遍历的节点的上一个节点。

进行中序遍历，每遍历到一个节点时，就将它的左孩子设为NULL，然后将它身作为上一个节点的右孩子。

struct TreeNode {

    int val;

    TreeNode \*left;

    TreeNode \*right;

    TreeNode(int x) : val(x), left(NULL), right(NULL) {}

};

class Solution {

public:

    TreeNode \*ans = new TreeNode(0),\*cur=ans;

    void inOrder(TreeNode\* node)

    {

        if(node==NULL)  return ;

        inOrder(node->left);

        node->left=NULL;    //将该节点的左孩子设为NULL

        cur->right=node;    //将该节点赋给上一个节点的右孩子

        cur=node;           //更新cur

        inOrder(node->right);

    }

    TreeNode\* convertBiNode(TreeNode\* root) {

        inOrder(root);

        return ans->right;

    }

};

[面试题 17.13. 恢复空格](https://leetcode-cn.com/problems/re-space-lcci/)

哦，不！你不小心把一个长篇文章中的空格、标点都删掉了，并且大写也弄成了小写。像句子"I reset the computer. It still didn’t boot!"已经变成了"iresetthecomputeritstilldidntboot"。在处理标点符号和大小写之前，你得先把它断成词语。当然了，你有一本厚厚的词典dictionary，不过，有些词没在词典里。假设文章用sentence表示，设计一个算法，把文章断开，要求未识别的字符最少，返回未识别的字符数。

**注意：**本题相对原题稍作改动，只需返回未识别的字符数

**示例：输入：**

dictionary = ["looked","just","like","her","brother"]

sentence = "jesslookedjustliketimherbrother"

**输出：** 7

**解释：** 断句后为"**jess** looked just like **tim** her brother"，共7个未识别字符。

class Trie {

public:

    Trie\* next[26] = {nullptr};

    bool isEnd;

    Trie() {

        isEnd = false;

    }

    void insert(string s) {

        Trie\* curPos = this;

        for (int i = s.length() - 1; i >= 0; --i) {

            int t = s[i] - 'a';

            if (curPos->next[t] == nullptr) {

                curPos->next[t] = new Trie();

            }

            curPos = curPos->next[t];

        }

        curPos->isEnd = true;

    }

};

class Solution {

public:

    int respace(vector<string>& dictionary, string sentence) {

        int n = sentence.length(), inf = 0x3f3f3f3f;

        Trie\* root = new Trie();

        for (auto& word: dictionary) {

            root->insert(word);

        }

        vector<int> dp(n + 1, inf);

        dp[0] = 0;

        for (int i = 1; i <= n; ++i) {

            dp[i] = dp[i - 1] + 1;

            Trie\* curPos = root;

            for (int j = i; j >= 1; --j) {

                int t = sentence[j - 1] - 'a';

                if (curPos->next[t] == nullptr) {

                    break;

                } else if (curPos->next[t]->isEnd) {

                    dp[i] = min(dp[i], dp[j - 1]);

                }

                if (dp[i] == 0) {

                    break;

                }

                curPos = curPos->next[t];

            }

        }

        return dp[n];

    }

};

[面试题 17.15. 最长单词](https://leetcode-cn.com/problems/longest-word-lcci/)

给定一组单词words，编写一个程序，找出其中的最长单词，且该单词由这组单词中的其他单词组合而成。若有多个长度相同的结果，返回其中字典序最小的一项，若没有符合要求的单词则返回空字符串。

**示例：**

**输入：** ["cat","banana","dog","nana","walk","walker","dogwalker"]

**输出：** "dogwalker"

**解释：** "dogwalker"可由"dog"和"walker"组成。

/\*

使用哈希集合存储列表中的每一个单词，接着遍历列表的中的每个单词判断其是否由其他单词组合而成

递归查找时，因为不能由其自身组成，因此每次从哈希集合中删除其自身

递归分割查找每一种可能，当一种分割到空串时，返回true，所有的情况遍历完毕，返回false

\*/

class Solution {

public:

    string longestWord(vector<string>& words) {

        unordered\_set<string> allwords(words.begin(),words.end());//可通過interator來複製vector到unordered\_set

        string ans;

        for(auto word:allwords){

            auto temp=allwords;//通過auto複製集合

            temp.erase(word);//erase()傳入元素可擦去集合中元素

            if(isCombinated(word,temp)){//查找word是否是其他元素組合

                if(word.size()>ans.size()) ans=word;

                if(word.size()==ans.size()) ans=min(word,ans);//Min傳入string,同樣長度情況下。min返回字典序小的那個string，實現返回通長度下字典序較小元素

            }

        }

        return ans;

    }

    bool isCombinated(string word,unordered\_set<string>& words){

        if(word.size()==0) return true;//迭代終點

        for(int i=1;i<=word.size();i++){//substr記錄首位不記錄末尾終點，所以i取等號為迭代末尾

        if(words.count(word.substr(0,i))&&isCombinated(word.substr(i), words)) return true;//words.count()查看元素是否在集合内，如果不在返回0

        }

        return false;

    }

};

[面试题 17.16. 按摩师](https://leetcode-cn.com/problems/the-masseuse-lcci/)

一个有名的按摩师会收到源源不断的预约请求，每个预约都可以选择接或不接。在每次预约服务之间要有休息时间，因此她不能接受相邻的预约。给定一个预约请求序列，替按摩师找到最优的预约集合（总预约时间最长），返回总的分钟数。

**注意：**本题相对原题稍作改动

**示例 1：输入：** [1,2,3,1]

**输出：** 4

**解释：** 选择 1 号预约和 3 号预约，总时长 = 1 + 3 = 4。

**示例 2：输入：** [2,7,9,3,1]

**输出：** 12

**解释：** 选择 1 号预约、 3 号预约和 5 号预约，总时长 = 2 + 9 + 1 = 12。

**示例 3：输入：** [2,1,4,5,3,1,1,3]

**输出：** 12

**解释：** 选择 1 号预约、 3 号预约、 5 号预约和 8 号预约，总时长 = 2 + 4 + 3 + 3 = 12。

class Solution {

public:

    int massage(vector<int>& nums) {

        int n = (int)nums.size();

        if (!n) return 0;

        int dp0 = 0, dp1 = nums[0];

        for (int i = 1; i < n; ++i){

            int tdp0 = max(dp0, dp1); // 计算 dp[i][0]

            int tdp1 = dp0 + nums[i]; // 计算 dp[i][1]

            dp0 = tdp0; // 用 dp[i][0] 更新 dp\_0

            dp1 = tdp1; // 用 dp[i][1] 更新 dp\_1

        }

        return max(dp0, dp1);

    }

};

[面试题 17.17. 多次搜索](https://leetcode-cn.com/problems/multi-search-lcci/)

给定一个较长字符串big和一个包含较短字符串的数组smalls，设计一个方法，根据smalls中的每一个较短字符串，对big进行搜索。输出smalls中的字符串在big里出现的所有位置positions，其中positions[i]为smalls[i]出现的所有位置。

**示例：**

**输入：**

big = "mississippi"

smalls = ["is","ppi","hi","sis","i","ssippi"]

**输出：** [[1,4],[8],[],[3],[1,4,7,10],[5]]

对smalls建trie树，其中每个树节点的sid记录对应的smalls id。

遍历big的所有后缀，并在trie树中查找后缀。对于查找路径上经过的所有有效sid（sid有效值为大于等于0的数），将后缀的起始id加入到sid对应的ans中。

struct TrieNode{

    int sid;

    TrieNode \*child[26];

    TrieNode(){

        sid=-1;

        for(int i=0;i<26;++i) child[i]=NULL;

    }

};

class Solution {

private:

    TrieNode \*root=new TrieNode();

public:

    void insert(string word,int s){

        int n=word.size();

        TrieNode \*cur=root;

        for(int i=0;i<n;++i){

            int cid=word.at(i)-'a';

            if(cur->child[cid]==NULL) cur->child[cid]=new TrieNode();

            cur=cur->child[cid];

        }

        cur->sid=s;

    }

    void search(string word,vector<vector<int>>& ans,int bid){

        int n=word.size();

        TrieNode \*cur=root;

        for(int i=0;i<n;++i){

            int cid=word.at(i)-'a';

            if(cur->sid!=-1) ans[cur->sid].push\_back(bid);

            if(cur->child[cid]==NULL) return ;

            cur=cur->child[cid];

        }

        if(cur->sid!=-1) ans[cur->sid].push\_back(bid);

    }

    vector<vector<int>> multiSearch(string big, vector<string>& smalls) {

        int n=smalls.size(),m=big.size();

        vector<vector<int>> ans(n,vector<int>{});

        for(int i=0;i<n;++i){

            if(smalls[i].size()==0) continue;

            insert(smalls[i],i);

        }

        for(int i=0;i<m;++i){

            string word=big.substr(i,m-i);

            search(word,ans,i);

        }

        return ans;

    }

};

[面试题 17.18. 最短超串](https://leetcode-cn.com/problems/shortest-supersequence-lcci/)

假设你有两个数组，一个长一个短，短的元素均不相同。找到长数组中包含短数组所有的元素的最短子数组，其出现顺序无关紧要。

返回最短子数组的左端点和右端点，如有多个满足条件的子数组，返回左端点最小的一个。若不存在，返回空数组。

**示例 1:输入:**

big = [7,5,9,0,2,1,3,**5,7,9,1**,1,5,8,8,9,7]

small = [1,5,9]

**输出:** [7,10]

**示例 2:输入:**

big = [1,2,3]

small = [4]

**输出:** []

class Solution {

public:

    vector<int> shortestSeq(vector<int>& big, vector<int>& small) {

        vector<int> res;

        unordered\_map<int, int> hashSmall;

        int count = 0, j = 0;

        for (auto e : small) {

            if (!hashSmall.count(e)) count++; // count的作用是记录small中元素的个数

            hashSmall[e]++;

        }

        for (int i = 0; i < big.size(); i++) {

            // 这一步的作用在于逐步扩大滑窗直至将所有的small元素包含进来

            hashSmall[big[i]]--;

            if (hashSmall[big[i]] == 0) count--;

            // count为0时进入循环

            while (!count) {

                // 由于上次处理后hashSmall中对应于big中非small部分hash的value = -1，因此，其加一后为0，循环直至找到第一个属于small的值，

                // 一次处理完成

                hashSmall[big[j]]++; // 这一步不仅用于分辨是否为small中元素，还用于找到下一个滑窗的进入位置。

                if (hashSmall[big[j]] > 0) {

                    count++;

                    if (res.empty() || res[1] - res[0] > i - j) res = {j, i};

                }

                j ++;

            }

        }

        return res;

    }

};

[面试题 17.19. 消失的两个数字](https://leetcode-cn.com/problems/missing-two-lcci/)

给定一个数组，包含从 1 到 N 所有的整数，但其中缺了两个数字。你能在 O(N) 时间内只用 O(1) 的空间找到它们吗？

以任意顺序返回这两个数字均可。

**示例 1:**

**输入:** [1]

**输出:** [2,3]

**示例 2:**

**输入:** [2,3]

**输出:** [1,4]

求和：

找到缺失的一个数

->找到缺失的两个数

class Solution {

public:

    vector<int> missingTwo(vector<int>& nums) {

        int n = nums.size() + 2;

        long sum = 0;

        for (auto x: nums) sum += x;

        int sumTwo = n \* (n + 1) / 2 - sum, limits = sumTwo / 2;

        sum = 0;

        for (auto x: nums)

            if (x <= limits) sum += x; // 两个数不相同那么一个大于，一个小于

        int one = limits \* (limits + 1) / 2 - sum;

        return {one, sumTwo - one};

    }

};

异或：

找到缺失的一个数

-> 找到缺失的两个数

class Solution {

public:

    vector<int> missingTwo(vector<int>& nums) {

        int ans = 0, n = nums.size();

        for (int i = 1; i <= n + 2; i ++) ans ^= i;

        for (auto x: nums) ans ^= x;

        int one = 0;

        int diff = ans & -ans;

        for (int i = 1; i <= n + 2; i ++)

            if (diff & i) one ^= i; // ?

        for (auto x: nums)

            if (diff & x) one ^= x;

        return {one, one ^ ans};

    }

};

原地hash

hash找到原来的位置

找到缺失的一个数

-> 找到缺失的两个数

class Solution {

public:

    vector<int> missingTwo(vector<int>& nums) {

        for (int i = 0; i < 3; i ++) nums.push\_back(-1);

        for (int i = 0; i < nums.size(); i ++)

            while (i != nums[i] && nums[i] != -1)

                swap(nums[i], nums[nums[i]]);

        vector<int> ans;

        for (int i = 1; i < nums.size(); i ++)

            if (nums[i] == -1) ans.push\_back(i);

        return ans;

    }

};

[面试题 17.21. 直方图的水量](https://leetcode-cn.com/problems/volume-of-histogram-lcci/)

给定一个直方图(也称柱状图)，假设有人从上面源源不断地倒水，最后直方图能存多少水量?直方图的宽度为 1。



上面是由数组 [0,1,0,2,1,0,1,3,2,1,2,1] 表示的直方图，在这种情况下，可以接 6 个单位的水（蓝色部分表示水）。

**示例:** **输入:** [0,1,0,2,1,0,1,3,2,1,2,1] **输出:** 6

对于每一列，找出当前列左边的最大值和右边的最大值，如果 curHeight < min(leftMax, rightMax)，则当前列水量为min(leftMax, rightMax) - curHeight，否则，当前不可能盛水。因此，首先需要构建左边和右边最大值的数组。实际上，左边最大值数组不需要提前构建，也不用数组，在求水量的过程中动态更新即可。

class Solution {

   public:

    int trap(vector<int>& height) {

        if (height.size() <= 2) return 0;

        int n = height.size();

        vector<int> rightMax(height.size());

        rightMax[n - 1] = 0;

        for (int i = n - 2; i >= 0; i--) {

            rightMax[i] = max(rightMax[i + 1], height[i + 1]);

        }

        int leftMax = 0;

        int res = 0;

        for (int i = 1; i < n - 1; i++) {

            leftMax = max(leftMax, height[i - 1]);

            int lowBound = min(leftMax, rightMax[i]);

            if (height[i] < lowBound) {

                res += (lowBound - height[i]);

            }

        }

        return res;

    }

};

[面试题 17.22. 单词转换](https://leetcode-cn.com/problems/word-transformer-lcci/)

给定字典中的两个词，长度相等。写一个方法，把一个词转换成另一个词， 但是一次只能改变一个字符。每一步得到的新词都必须能在字典中找到。

编写一个程序，返回一个可能的转换序列。如有多个可能的转换序列，你可以返回任何一个。

**示例 1:** **输入:**

beginWord = "hit",

endWord = "cog",

wordList = ["hot","dot","dog","lot","log","cog"]

**输出:** ["hit","hot","dot","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。

class Solution {

public:

    bool canTranslate(string& from\_, string& to\_){

        if(from\_.size() != to\_.size()) { return false; }

        int count = 0;

        for(int i = 0; i < from\_.size(); ++i){

            if(from\_[i] != to\_[i]) {

                count++;

            }

        }

        return count == 1;

    }

    bool hasRoute(string& curWord, string& endWord, vector<string>& wordList,

                    vector<bool>& visited, vector<string>& result) {

        if(curWord == endWord){ return true; }

        for(int i = 0; i < wordList.size(); ++i){

            if(visited[i] || !canTranslate(curWord, wordList[i])) continue;

            visited[i] = true;

            result.push\_back(wordList[i]);

            if(hasRoute(wordList[i], endWord, wordList, visited, result)){

                return true;

            }

            result.pop\_back();

            // 如果运行到这一步，意味着无法从i这个点找到路径，所以visited[i]无需改为false.

            // visited[i] = false;

        }

        return false;

    }

    vector<string> findLadders(string beginWord, string endWord, vector<string>& wordList) {

        vector<string> result = {beginWord};

        vector<bool> visited(wordList.size(), false);

        if(hasRoute(beginWord, endWord, wordList, visited, result)){

            return result;

        }

        return vector<string>();

    }

};

[面试题 17.23. 最大黑方阵](https://leetcode-cn.com/problems/max-black-square-lcci/)

给定一个方阵，其中每个单元(像素)非黑即白。设计一个算法，找出 4 条边皆为黑色像素的最大子方阵。

返回一个数组 [r, c, size] ，其中 r, c 分别代表子方阵左上角的行号和列号，size 是子方阵的边长。若有多个满足条件的子方阵，返回 r 最小的，若 r 相同，返回 c 最小的子方阵。若无满足条件的子方阵，返回空数组。

**示例 1:输入:**

[

  [1,0,1],

  [**0,0**,1],

  [**0,0**,1]

]

**输出:** [1,0,2]

**解释:** 输入中 0 代表黑色，1 代表白色，标粗的元素即为满足条件的最大子方阵

**示例 2:**

**输入:**

[

  [**0**,1,1],

  [1,0,1],

  [1,1,0]

]

**输出:** [0,0,1]

动态规划，cnt[r][c][0/1]表示以坐标r,c为起点向左/右最多的连续黑色块的数量

class Solution {

public:

    vector<int> findSquare(vector<vector<int>>& matrix) {

        vector<int> ans(3, 0);

        int n = matrix.size();

        if(n == 0) return {};

        if(n == 1){

            if(matrix[0][0] == 0)

                return {0, 0, 1};

            else

                return {};

        }

        //cnt[r][c][0/1],0右侧,1下侧

        vector<vector<vector<int>>> cnt(n, vector<vector<int>>(n, vector<int>(2)));

        for(int r = n-1; r >= 0; r--){

            for(int c = n-1; c >= 0; c--){

                if(matrix[r][c] == 1)

                    cnt[r][c][0] = cnt[r][c][1] = 0;

                else{

                    //统计cnt[r][c][0/1]

                    if(r < n-1) cnt[r][c][1] = cnt[r+1][c][1] + 1;

                    else cnt[r][c][1] = 1;

                    if(c < n-1) cnt[r][c][0] = cnt[r][c+1][0] + 1;

                    else cnt[r][c][0] = 1;

                   //更新当前最大子方阵

                    int len = min(cnt[r][c][0], cnt[r][c][1]);//最大的可能的边长

                    while(len >= ans[2]){//要答案r,c最小，所以带等号

                        if(cnt[r+lena-1][c][0] >= len && cnt[r][c+len-1][1] >= len){

                            //可以构成长为len的方阵

                            ans = {r, c, len};

                            break;

                        }

                        len--;

                    }

                }

            }

        }

        return ans;

    }

};

[面试题 17.24. 最大子矩阵](https://leetcode-cn.com/problems/max-submatrix-lcci/)

给定一个正整数和负整数组成的 N × M 矩阵，编写代码找出元素总和最大的子矩阵。

返回一个数组 [r1, c1, r2, c2]，其中 r1, c1 分别代表子矩阵左上角的行号和列号，r2, c2 分别代表右下角的行号和列号。若有多个满足条件的子矩阵，返回任意一个均可。

**注意：**本题相对书上原题稍作改动

**示例:** **输入:**

[

  [-1,**0**],

  [0,-1]

]

**输出:** [0,1,0,1] **解释:** 输入中标粗的元素即为输出所表示的矩阵

class Solution {

public:

    vector<int> getMaxMatrix(vector<vector<int>>& matrix) {

        vector<int> ans(4);//保存最大子矩阵的左上角和右下角的行列坐标

        int N = matrix.size();

        int M = matrix[0].size();

        vector<int> b(M,0);//记录当前i~j行组成大矩阵的每一列的和，将二维转化为一维

        int sum;//相当于dp[i],dp\_i

        int maxsum=INT\_MIN;//记录最大值

        int bestr1,bestc1;//暂时记录左上角，相当于begin

        for(int i=0;i<N;i++){     //以i为上边，从上而下扫描

            for(int t=0;t<M;t++ ) b[t]=0;    //每次更换子矩形上边，就要清空b，重新计算每列的和

            for(int j=i;j<N;j++){    //子矩阵的下边，从i到N-1，不断增加子矩阵的高

                //一下就相当于求一次最大子序列和

                sum = 0;//从头开始求dp

                for(int k=0;k<M;k++){

                    b[k]+=matrix[j][k];

//我们只是不断增加其高，也就是下移矩阵下边，所有这个矩阵每列的和只需要加上新加的哪一行的元素

//因为我们求dp[i]的时候只需要dp[i-1]和nums[i],所有在我们不断更新b数组时就可以求出当前位置的dp\_i

                    if(sum>0){

                        sum+=b[k];

                    }

                    else{

                        sum=b[k];

                        bestr1=i;//自立门户，暂时保存其左上角

                        bestc1=k;

                    }

                    if( sum > maxsum){

                        maxsum = sum;

                        ans[0]=bestr1;//更新答案

                        ans[1]=bestc1;

                        ans[2]=j;

                        ans[3]=k;

                    }

                }

            }

        }

        return ans;

    }

};

[面试题 17.25. 单词矩阵](https://leetcode-cn.com/problems/word-rectangle-lcci/)

给定一份单词的清单，设计一个算法，创建由字母组成的面积最大的矩形，其中每一行组成一个单词(自左向右)，每一列也组成一个单词(自上而下)。不要求这些单词在清单里连续出现，但要求所有行等长，所有列等高。

如果有多个面积最大的矩形，输出任意一个均可。一个单词可以重复使用。

**示例 1:**

**输入:** ["this", "real", "hard", "trh", "hea", "iar", "sld"]

**输出:**

[

  "this",

  "real",

  "hard"

]

**示例 2:**

**输入:** ["aa"]

**输出:** ["aa","aa"]

将所有单词插入Trie树

将单词按长度分组，哈希map

从单词长度最长组的开始遍历，对每组单词进行DFS搜索

利用Trie树检查是否合法，不合法回溯

有几处优化见注释，容易超时

class trie

{

public:

    bool isEnd = false;

    trie\* next[26] = {NULL};

    void insert(string& s)

    {

        trie \*cur = this;

        for(int i = 0; i < s.size(); ++i)

        {

            if(!cur->next[s[i]-'a'])

                cur->next[s[i]-'a'] = new trie();

            cur = cur->next[s[i]-'a'];

        }

        cur->isEnd = true;

    }

};

class Solution {

    trie\* t;

    vector<string> ans;

    vector<string> temp;

public:

    vector<string> maxRectangle(vector<string>& words) {

        t = new trie();

        map<int,vector<string>> m;

        int maxlen = 0, maxarea = 0, area;

        for(auto& w : words)

        {

            t->insert(w);//单词插入trie

            m[w.size()].push\_back(w);//单词按长度分组

            maxlen = max(maxlen, int(w.size()));//最大单词长度

        }

        for(auto it = m.rbegin(); it != m.rend(); ++it)

        {   //反向遍历，从长度大的开始

            if(maxarea/(it->first) >= maxlen)

                break;//最长的单词\*宽度 都不够大，这组不用找了

            temp.clear();

            area = 0;

            dfs(it->second,maxarea,maxlen,area);

        }

        return ans;

    }

    void dfs(vector<string>& wd, int& maxarea, int maxlen, int area)

    {

        if(wd[0].size()\*maxlen <= maxarea)//找到的面积到极限了，退出吧

            return;//这个优化必须有，没有会超时

        vector<bool> res;

        for(int i = 0; i < wd.size(); ++i)

        {

            temp.push\_back(wd[i]);

            res = isgood(temp);//是否合法

            if(res[0])//可以往下加单词

            {

                area = temp.size()\*temp[0].size();

                if(res[1] && area > maxarea)//都是结束单词

                {   //更新最大值

                    maxarea = area;

                    ans = temp;

                }

                dfs(wd, maxarea, maxlen, area);

            }

            // else//不能有else，有的话 if内的dfs出来，没有回溯

            temp.pop\_back();

        }

    }

    vector<bool> isgood(vector<string>& tp)

    {

        trie \*cur;

        bool allend = true;

        int i, j, n = tp[0].size();

        for(j = 0; j < n; ++j)//按列在trie中检查

        {

            cur = t;

            for(i = 0; i < tp.size(); ++i)

            {

                if(!cur->next[tp[i][j]-'a'])

                    return {false, false};

                cur = cur->next[tp[i][j]-'a'];

            }

            allend &= cur->isEnd;

        }

        return {true, allend};//可以继续插入、每个列向单词都在字典中

    }

};

[面试题 17.26. 稀疏相似度](https://leetcode-cn.com/problems/sparse-similarity-lcci/)

两个(具有不同单词的)文档的交集(intersection)中元素的个数除以并集(union)中元素的个数，就是这两个文档的相似度。例如，{1, 5, 3} 和 {1, 7, 2, 3} 的相似度是 0.4，其中，交集的元素有 2 个，并集的元素有 5 个。给定一系列的长篇文档，每个文档元素各不相同，并与一个 ID 相关联。它们的相似度非常“稀疏”，也就是说任选 2 个文档，相似度都很接近 0。请设计一个算法返回每对文档的 ID 及其相似度。只需输出相似度大于 0 的组合。请忽略空文档。为简单起见，可以假定每个文档由一个含有不同整数的数组表示。

输入为一个二维数组 docs，docs[i] 表示 id 为 i 的文档。返回一个数组，其中每个元素是一个字符串，代表每对相似度大于 0 的文档，其格式为 {id1},{id2}: {similarity}，其中 id1 为两个文档中较小的 id，similarity 为相似度，精确到小数点后 4 位。以任意顺序返回数组均可。

**示例:**

**输入:**

[

  [14, 15, 100, 9, 3],

  [32, 1, 9, 3, 5],

  [15, 29, 2, 6, 8, 7],

  [7, 10]

]

**输出:**

[

  "0,1: 0.2500",

  "0,2: 0.1000",

  "2,3: 0.1429"

]

基本思路：

先遍历所有文档的所有单词，mp1[word]返回数组，表示包含有单词word的文档id

生成mp1需要O(DW)复杂度，D表示文档数量，W表示每个文档的单词数（粗略认为W相等）。

根据mp1我们就可以判断哪两个文档会相交，只对相交的文档进行相似度计算。即使相似性的计算这块是个暴力法，速度也不会很慢，因为题目说了相交是稀疏的，只有很少的文档有会相交。

知道哪两个文档会相交后，分别排序两个文档内的单词，然后双指针找出两个文档相交元素，进一步的计算出这两个文档的相似度。

进一步提升效率

从判断相交的方法入手。通常的方法是把相交文档内的单词排序，然后双指针找出两个文档相交元素。但是我们可以改进方法，利用好mp1的信息，进一步提升效率。

对于遍历mp1种每个word，都可以得到mp1[word]，这是一个数组，返回包含有单词word的文档id

遍历mp1[word]，对两两文档的相交次数+1，即mp2[id1][id2]++;

最后mp2里存的是所有相交文档的交集个数

相似度(id1,id2) = mp2[id1][id2] / (docs[id1].size() + docs[id2].size() - mp2[id1][id2])

注意：C++注意精度误差问题，最后输出时需要加上1e-9

class Solution {

public:

    vector<string> computeSimilarities(vector<vector<int>>& docs) {

        unordered\_map<int, vector<int>> mp1;

        for(int i=0; i<docs.size(); ++i){

            for(auto &word : docs[i]){

                mp1[word].push\_back(i);

            }

        }

        unordered\_map<int, unordered\_map<int, int>> mp2;

        for(auto &item : mp1){

            auto &ids = item.second;

            for(int i=0; i+1<ids.size(); ++i){

                for(int j=i+1; j<ids.size(); ++j){

                    mp2[ids[i]][ids[j]]++;

                }

            }

        }

        vector<string> result;

        char temp[256];

        for(auto &item : mp2){

            int id1 = item.first;

            for(auto &item2 : item.second){

                int id2 = item2.first;

                double similary = (double)item2.second /

                                    (docs[id1].size() + docs[id2].size() - item2.second);

                sprintf(temp, "%d,%d: %0.4lf", id1, id2, similary + 1e-9);

                // cout << temp << endl; // debug

                result.push\_back(temp);

            }

        }

        return result;

    }

};