## [剑指 Offer 03. 数组中重复的数字](https://leetcode-cn.com/problems/shu-zu-zhong-zhong-fu-de-shu-zi-lcof/)

找出数组中重复的数字。

在一个长度为 n 的数组 nums 里的所有数字都在 0～n-1 的范围内。数组中某些数字是重复的，但不知道有几个数字重复了，也不知道每个数字重复了几次。请找出数组中任意一个重复的数字。

示例 1：

输入：

[2, 3, 1, 0, 2, 5, 3]

输出：2 或 3

限制：

2 <= n <= 100000

**解：**

排序（自己的解法）：

对输入数组进行排序，遍历数组，如果有重复数字相邻两个数字应该相等，利用这一性质找出重复数字。

时间O(NlogN)(排序O(logN)，遍历O(N)) 空间 O(1)

class Solution {

public:

    int findRepeatNumber(vector<int>& nums) {

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

        int res = -1;

        for(auto i : nums){

            if(res != i){

                res = i;

            }

            else{

                return res;

            }

        }

        return res;

    }

};

原地置换：

题目说明尚未被充分使用，即在一个长度为 n 的数组 nums 里的所有数字都在 0 ~ n-1的范围内 。此说明含义：数组元素的索引和值是一对多的关系。因此，可遍历数组并通过交换操作，使元素的索引与值 一一对应（即nums[i] = i）。因而，就能通过索引映射对应的值，起到与字典等价的作用。

遍历中，第一次遇到数字 x 时，将其交换至索引 x 处；而当第二次遇到数字 x 时，一定有 nums[x] = x ，此时即可得到一组重复数字。

时间O(N) 空间 O(1)

class Solution {

public:

    int findRepeatNumber(vector<int>& nums) {

        int i = 0;

        while(i < nums.size()) {

            if(nums[i] == i) {

                i++;

                continue;

            }

            if(nums[nums[i]] == nums[i])

                return nums[i];

            swap(nums[i],nums[nums[i]]);

        }

        return -1;

    }

};

其他解法：利用hash表存储统计是否重复出现。

时间O(N) 空间O(N)

## [剑指 Offer 04. 二维数组中的查找](https://leetcode-cn.com/problems/er-wei-shu-zu-zhong-de-cha-zhao-lcof/)

在一个 n \* m 的二维数组中，每一行都按照从左到右递增的顺序排序，每一列都按照从上到下递增的顺序排序。请完成一个高效的函数，输入这样的一个二维数组和一个整数，判断数组中是否含有该整数。

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。

限制：

0 <= n <= 1000

0 <= m <= 1000

**解：**

暴力遍历：遍历该二维数组直到找到该数字。

时间O(nm) 空间 O(1)

线性查找（自己的解法）：

由于二维数组每一行都按照从左到右递增的顺序排序，每一列都按照从上到下递增的顺序排序，我们可以从左下角开始。若目标大于当前值，则列++，若目标小与当前值，则行--，直到找到目标值或到达数组边界。

时间O(n+m) 空间 O(1)

class Solution {

public:

    bool findNumberIn2DArray(vector<vector<int>>& matrix, int target) {

        int row = matrix.size()-1;

        int col = 0;

        while(row > -1 && col < matrix[0].size()){

            if(target < matrix[row][col])

                row--;

            else if(target > matrix[row][col])

                col++;

            else return true;

        }

        return false;

    }

};

## [剑指 Offer 05. 替换空格](https://leetcode-cn.com/problems/ti-huan-kong-ge-lcof/)

请实现一个函数，把字符串 s 中的每个空格替换成"%20"。

示例 1：

输入：s = "We are happy."

输出："We%20are%20happy."

限制：

0 <= s 的长度 <= 10000

**解：**

自己的解法：

创建一个新字符串用来保存结果，遍历原始字符串，若不是空格就将字符加入新字符串，若是空格则将”%20”加入。

时间O(N) 空间O(N)

class Solution {

public:

    string replaceSpace(string s) {

        string str ;

        for(auto& i : s){

            if(i == ' '){

                str.push\_back('%');

                str.push\_back('2');

                str.push\_back('0');

            }else

            {

                str.push\_back(i);

            }

        }

        return str;

    }

};

先扩充后倒序填充：

第一次遍历统计出空格的数量n并原地扩展2n的空间以足够存放扩充后的字符串。第二次倒序遍历，遇到字符就直接填充，遇到空格就将”%20”填入。

时间O(N) 空间O(1)

class Solution {

public:

    string replaceSpace(string s) {

            if(s.empty()){

                return s;

            }

            int length = s.length() - 1;

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

                if(s[i] == ' '){

                    s +="00";

                }

            }

            int length2 = s.length() - 1;

            while(length >= 0 && length2 >= length){

                if(s[length] == ' '){

                    s[length2 -- ] = '0';

                    s[length2 -- ] = '2';

                    s[length2 -- ] = '%';

                }

                else{

                    s[length2 -- ] = s[length];

                }

                length --;

            }

            return s;

    }

};

## [剑指 Offer 06. 从尾到头打印链表](https://leetcode-cn.com/problems/cong-wei-dao-tou-da-yin-lian-biao-lcof/)

输入一个链表的头节点，从尾到头反过来返回每个节点的值（用数组返回）。

示例 1：

输入：head = [1,3,2]

输出：[2,3,1]

限制：

0 <= 链表长度 <= 10000

**解：**

翻转法：

建立一个数组用来保存结果，遍历链表将节点的值逐一加入，最后将数组反转。

时间O(N) 空间O(N)

/\*\*

 \* Definition for singly-linked list.

 \* struct ListNode {

 \*     int val;

 \*     ListNode \*next;

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

 \* };

 \*/

class Solution {

public:

    vector<int> reversePrint(ListNode\* head) {

        vector<int> res;

        ListNode\* p = head;

        while(p){

            res.push\_back(p->val);

            p = p->next;

        }

        reverse(res.begin(),res.end());

        return res;

    }

};

辅助栈法：

用栈存储遍历链表节点的值，逐一出栈存入结果数组。

时间O(N) 空间O(N)

递归法：

class Solution {

public:

    vector<int> res;

    vector<int> reversePrint(ListNode\* head) {

        if(head == nullptr)

            return res;

        reversePrint(head->next);

        res.push\_back(head->val);

        return res;

    }

};

效率较低。

## [剑指 Offer 07. 重建二叉树](https://leetcode-cn.com/problems/zhong-jian-er-cha-shu-lcof/)

输入某二叉树的前序遍历和中序遍历的结果，请重建该二叉树。假设输入的前序遍历和中序遍历的结果中都不含重复的数字。

例如，给出

前序遍历 preorder = [3,9,20,15,7]

中序遍历 inorder = [9,3,15,20,7]

返回如下的二叉树：

3

/ \

9 20

/ \

15 7

限制：

0 <= 节点个数 <= 5000

**解：**

递归求解（自己的解法）：

对于任意一颗树而言，前序遍历的形式总是：

[ 根节点, [左子树的前序遍历结果], [右子树的前序遍历结果] ]

即根节点总是前序遍历中的第一个节点。而中序遍历的形式总是：

[ [左子树的中序遍历结果], 根节点, [右子树的中序遍历结果] ]

只要我们在中序遍历中定位到根节点，那么我们就可以分别知道左子树和右子树中的节点数目。由于同一颗子树的前序遍历和中序遍历的长度显然是相同的，因此我们就可以对应到前序遍历的结果中，对上述形式中的所有左右括号进行定位。

这样以来，我们就知道了左子树的前序遍历和中序遍历结果，以及右子树的前序遍历和中序遍历结果，我们就可以递归地对构造出左子树和右子树，再将这两颗子树接到根节点的左右位置。

/\*\*

 \* Definition for a binary tree node.

 \* struct TreeNode {

 \*     int val;

 \*     TreeNode \*left;

 \*     TreeNode \*right;

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

 \* };

 \*/

class Solution {

public:

    int get\_position(int value,vector<int>& v){

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

            if(v[i] == value){

                return i;

            }

        }

        return -1;

    }

    TreeNode\* root = new TreeNode(0);

    TreeNode\* buildTree(vector<int>& preorder, vector<int>& inorder) {

        if(preorder.size() == 0){

            return 0;

        }

        int rt = preorder[0];

        if(root->val == 0){

            root->val = rt;

        }

        TreeNode\* res = root;

        int second = 0;

        int first = 0;

        int third = preorder.size()-1;

        second = get\_position(rt,inorder);

        if(second - first == 1){

            root->left = new TreeNode(inorder[first]);

        }

        else if(second > first){

            vector<int> preo,ino;

            for(int i = first; i < second; ++i){

                preo.push\_back(preorder[i+1]);

                ino.push\_back(inorder[i]);

            }

            int r = preo[0];

            TreeNode\* n = new TreeNode(r);

            root->left = n;

            root = root->left;

            buildTree(preo,ino);

        }//对左子树进行递归

        root = res;//左子树结束回到根节点

        if(third - second == 1){

            root->right = new TreeNode(inorder[third]);

        }

        else if(second < third){

            vector<int> preo,ino;

            for(int i = second+1; i <= third; ++i){

                preo.push\_back(preorder[i]);

                ino.push\_back(inorder[i]);

            }

            int r = preo[0];

            TreeNode\* n = new TreeNode(r);

            root->right = n;

            root = root->right;

            buildTree(preo,ino);

        }//对右子树进行递归

        return res;

    }

};

递归优化版：

在中序遍历中对根节点进行定位时，一种简单的方法是直接扫描整个中序遍历的结果并找出根节点，但这样做的时间复杂度较高。我们可以考虑使用哈希表来帮助我们快速地定位根节点。对于哈希映射中的每个键值对，键表示一个元素（节点的值），值表示其在中序遍历中的出现位置。在构造二叉树的过程之前，我们可以对中序遍历的列表进行一遍扫描，就可以构造出这个哈希映射。在此后构造二叉树的过程中，我们就只需要 O(1)O(1) 的时间对根节点进行定位了。

时间O(N) 空间O(N)

class Solution {

public:

    vector<int> Preorder ;

    unordered\_map<int,int> dic;

    TreeNode\* build(int pre\_root ,int in\_left ,int in\_right){

        //如果左边界大于右边界说明到过了叶子

        if(in\_left > in\_right){

            return NULL;

        }

        //pre\_root 是先序里面的索引 ！！

        TreeNode\* root = new TreeNode(Preorder[pre\_root]);

        //获取先序中的节点在中序中的节点， 即index 左边就是这节点的左子树，index右边就是节点的右子树

        int index = dic[Preorder[pre\_root]];

        //当前节点左树即为先序索引+1 （没了话会在下一次迭代返回NULL）

        root->left = build(pre\_root+1,in\_left,index-1);

        //当前节点右树即为 根结点在前序中的索引+左树所有节点数（即节点在中序中的索引）-左边界+1 ，下一次的左边界为根在中序的索引+1

        root->right = build(pre\_root+index-in\_left+1,index+1 ,in\_right);

        return root;

    }

    TreeNode\* buildTree(vector<int>& preorder, vector<int>& inorder) {

        //赋值至外部变量

        Preorder = preorder;

        //使用map映射inorder的值和索引，提高找到索引效率

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

            dic[inorder[i]] = i;

        }

        return build(0,0,preorder.size()-1);

    }

};

迭代版本：

我们用一个栈和一个指针辅助进行二叉树的构造。初始时栈中存放了根节点（前序遍历的第一个节点），指针指向中序遍历的第一个节点；

我们依次枚举前序遍历中除了第一个节点以外的每个节点。如果 index 恰好指向栈顶节点，那么我们不断地弹出栈顶节点并向右移动 index，并将当前节点作为最后一个弹出的节点的右儿子；如果 index 和栈顶节点不同，我们将当前节点作为栈顶节点的左儿子；

无论是哪一种情况，我们最后都将当前的节点入栈。

最后得到的二叉树即为答案。

时间O(N) 空间O(N)

class Solution {

public:

    TreeNode\* buildTree(vector<int>& preorder, vector<int>& inorder) {

        if (!preorder.size()) {

            return nullptr;

        }

        TreeNode\* root = new TreeNode(preorder[0]);

        stack<TreeNode\*> stk;

        stk.push(root);

        int inorderIndex = 0;

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

            int preorderVal = preorder[i];

            TreeNode\* node = stk.top();

            if (node->val != inorder[inorderIndex]) {

                node->left = new TreeNode(preorderVal);

                stk.push(node->left);

            }

            else {

                while (!stk.empty() && stk.top()->val == inorder[inorderIndex]) {

                    node = stk.top();

                    stk.pop();

                    ++inorderIndex;

                }

                node->right = new TreeNode(preorderVal);

                stk.push(node->right);

            }

        }

        return root;

    }

};

## [剑指 Offer 09. 用两个栈实现队列](https://leetcode-cn.com/problems/yong-liang-ge-zhan-shi-xian-dui-lie-lcof/)

用两个栈实现一个队列。队列的声明如下，请实现它的两个函数 appendTail 和 deleteHead ，分别完成在队列尾部插入整数和在队列头部删除整数的功能。(若队列中没有元素，deleteHead 操作返回 -1 )

示例 1：

输入：

["CQueue","appendTail","deleteHead","deleteHead"]

[[],[3],[],[]]

输出：[null,null,3,-1]

示例 2：

输入：

["CQueue","deleteHead","appendTail","appendTail","deleteHead","deleteHead"]

[[],[],[5],[2],[],[]]

输出：[null,-1,null,null,5,2]

提示：

1 <= values <= 10000

最多会对 appendTail、deleteHead 进行 10000 次调用

**解：**

双辅助栈（自己的解法）：

建立栈s1，s2。

对于Cqueue，将两个栈清空。

对于appendTail，将值推入s1中。

对于deleteHead，将s2用作输出，若s2不空，将s2的top作为输出。当s2空，若s1不空，则将s1中元素转移到s2（倒置，以实现队列的效果）。若s1，s2都空，则返回-1。

class CQueue {

    stack<int> s1,s2;

public:

    CQueue() {

        while(!s1.empty()){

            s1.pop();

        }

        while(!s2.empty()){

            s1.pop();

        }

    }

    void appendTail(int value) {

        s1.push(value);

    }

    int deleteHead() {

        if(!s2.empty()){

            int res = 0;

            res = s2.top();

            s2.pop();

            return res;

        }

        else if(!s1.empty()){

            int res;

            adjust();

            res = deleteHead();

            return res;

        }

        else{

            return -1;

        }

    }

    void adjust(){

        int tmp;

        while(!s1.empty()){

            tmp = s1.top();

            s1.pop();

            s2.push(tmp);

        }

    }

};

## [剑指 Offer 10- I. 斐波那契数列](https://leetcode-cn.com/problems/fei-bo-na-qi-shu-lie-lcof/)

写一个函数，输入 n ，求斐波那契（Fibonacci）数列的第 n 项（即 F(N)）。斐波那契数列的定义如下：

F(0) = 0,   F(1) = 1

F(N) = F(N - 1) + F(N - 2), 其中 N > 1.

斐波那契数列由 0 和 1 开始，之后的斐波那契数就是由之前的两数相加而得出。

答案需要取模 1e9+7（1000000007），如计算初始结果为：1000000008，请返回 1。

示例 1：

输入：n = 2

输出：1

示例 2：

输入：n = 5

输出：5

提示：

0 <= n <= 100

**解：**

动态规划：

dp[i]=dp[i-1]+dp[i-2]

时间O(n) 空间O(n)

class Solution {

public:

    int fib(int n) {

        /\*动态规划 dp[i]=dp[i-1]+dp[i-2]

        \*/

        if(n<0) return -1;

        if(n==0) return 0;

        if(n==1) return 1;

        else{

            int dp[n+1];

            dp[0]=0; dp[1]=1;

            for(int i=2;i<n+1;i++){

                dp[i]=(dp[i-1]+dp[i-2])%1000000007;

            }

            return dp[n];

        }

    }

};

迭代（自己的解法）：

时间O(n) 空间O(1)

class Solution {

public:

    int fib(int n) {

        if(n<0) return -1;

        if(n==0) return 0;

        if(n==1) return 1;

        else{

            int res = 0;

            int t1 = 0;

            int t2 = 1;

            for(int i=2;i<n+1;i++){

                res = (t1 + t2)%1000000007;

                t1 = t2;

                t2 = res;

            }

            return res;

        }

    }

};

## [剑指 Offer 10- II. 青蛙跳台阶问题](https://leetcode-cn.com/problems/qing-wa-tiao-tai-jie-wen-ti-lcof/)

一只青蛙一次可以跳上1级台阶，也可以跳上2级台阶。求该青蛙跳上一个 n 级的台阶总共有多少种跳法。

答案需要取模 1e9+7（1000000007），如计算初始结果为：1000000008，请返回 1。

示例 1：

输入：n = 2

输出：2

示例 2：

输入：n = 7

输出：21

示例 3：

输入：n = 0

输出：1

提示：

0 <= n <= 100

**解：**

设跳上 n 级台阶有 f(n) 种跳法。在所有跳法中，青蛙的最后一步只有两种情况： 跳上 1 级或 2 级台阶。

当为 1 级台阶： 剩 n-1 个台阶，此情况共有 f(n-1) 种跳法；

当为 2 级台阶： 剩 n-2 个台阶，此情况共有 f(n-2) 种跳法。

f(n) 为以上两种情况之和，即 f(n)=f(n-1)+f(n-2) ，以上递推性质为斐波那契数列。本题可转化为求斐波那契数列第 nn 项的值，与面试题10- I.斐波那契数列等价，唯一的不同在于起始数字不同。

青蛙跳台阶问题： f(0)=1f(0)=1 , f(1)=1f(1)=1 , f(2)=2f(2)=2 ；

斐波那契数列问题： f(0)=0f(0)=0 , f(1)=1f(1)=1 , f(2)=1f(2)=1 。

迭代（自己的解法）：

时间O(n) 空间O(1)

class Solution {

public:

    int numWays(int n) {

        int res = 0;

        if(n==0){

            return 1;

        }

        else if(n == 1){

            return 1;

        }

        else if(n == 2){

            return 2;

        }

        else{

            int r1 = 1;

            int r2 = 2;

            for(int i = 3; i <= n; ++i){

                res = (r1 + r2)%1000000007;

                r1 = r2;

                r2 = res;

            }

            return res %1000000007;

        }

    }

};

## [剑指 Offer 11. 旋转数组的最小数字](https://leetcode-cn.com/problems/xuan-zhuan-shu-zu-de-zui-xiao-shu-zi-lcof/)

把一个数组最开始的若干个元素搬到数组的末尾，我们称之为数组的旋转。输入一个递增排序的数组的一个旋转，输出旋转数组的最小元素。例如，数组 [3,4,5,1,2] 为 [1,2,3,4,5] 的一个旋转，该数组的最小值为1。

示例 1：

输入：[3,4,5,1,2]

输出：1

示例 2：

输入：[2,2,2,0,1]

输出：0

**解：**

遍历：

遍历数组找出最小值。

时间O(n) 空间O(1)

二分查找（自己的解法）：

初始化： 声明 i, j 双指针分别指向 nums 数组左右两端；

循环二分： 设 m = (i + j) / 2 为每次二分的中点（ "/" 代表向下取整除法，因此恒有  i≤m<j  ），可分为以下三种情况：

当 nums[m] > nums[j] 时： m 一定在左排序数组中，即旋转点 x 一定在 [m + 1, j] 闭区间内，因此执行 i = m + 1；

当 nums[m] < nums[j] 时： m 一定在右排序数组中，即旋转点 x 一定在[i, m] 闭区间内，因此执行 j = m；

当 nums[m] = nums[j] 时： 无法判断 m 在哪个排序数组中，即无法判断旋转点 x 在 [i, m] 还是 [m + 1, j] 区间中。解决方案： 执行 j = j - 1 缩小判断范围。

返回值： 当 i = j时跳出二分循环，并返回 旋转点的值 nums[i] 即可。

时间O(logn) 空间O(1)

class Solution {

public:

    int minArray(vector<int>& numbers) {

        int low = 0;

        int high = numbers.size()-1;

        while(low < high){

            int mid = (low + high) / 2 ;

            if(numbers[mid] > numbers[high]){

                low = mid + 1;

            }

            else if(numbers[mid] < numbers[high]){

                high = mid;

            }

            else{

                high--;

            }

        }

        return numbers[low];

    }

};

## [剑指 Offer 12. 矩阵中的路径](https://leetcode-cn.com/problems/ju-zhen-zhong-de-lu-jing-lcof/)

请设计一个函数，用来判断在一个矩阵中是否存在一条包含某字符串所有字符的路径。路径可以从矩阵中的任意一格开始，每一步可以在矩阵中向左、右、上、下移动一格。如果一条路径经过了矩阵的某一格，那么该路径不能再次进入该格子。例如，在下面的3×4的矩阵中包含一条字符串“bfce”的路径（路径中的字母用加粗标出）。

[["a","b","c","e"],

["s","f","c","s"],

["a","d","e","e"]]

但矩阵中不包含字符串“abfb”的路径，因为字符串的第一个字符b占据了矩阵中的第一行第二个格子之后，路径不能再次进入这个格子。

示例 1：

输入：board = [["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]], word = "ABCCED"

输出：true

示例 2：

输入：board = [["a","b"],["c","d"]], word = "abcd"

输出：false

提示：

1 <= board.length <= 200

1 <= board[i].length <= 200

**解：**

DFS+剪枝（递归）：

深度优先搜索： 可以理解为暴力法遍历矩阵中所有字符串可能性。DFS 通过递归，先朝一个方向搜到底，再回溯至上个节点，沿另一个方向搜索，以此类推。

剪枝： 在搜索中，遇到 这条路不可能和目标字符串匹配成功 的情况（例如：此矩阵元素和目标字符不同、此元素已被访问），则应立即返回，称之为 可行性剪枝 。

DFS 解析：

递归参数： 当前元素在矩阵 board 中的行列索引 i 和 j ，当前目标字符在 word 中的索引 k 。

终止条件：

返回 false ： (1) 行或列索引越界 (2) 当前矩阵元素与目标字符不同 (3) 当前矩阵元素已访问过 （ (3) 可合并至 (2) ） 。

返回 true ： k = len(word) - 1 ，即字符串 word 已全部匹配。

递推工作：

标记当前矩阵元素： 将 board[i][j] 修改为空字符，代表此元素已访问过，防止之后搜索时重复访问。

搜索下一单元格： 朝当前元素的 上、下、左、右 四个方向开启下层递归，使用 或 连接 （代表只需找到一条可行路径就直接返回，不再做后续 DFS ），并记录结果至 res 。

还原当前矩阵元素： 将 board[i][j] 元素还原至初始值，即 word[k] 。

返回值： 返回布尔量 res ，代表是否搜索到目标字符串。

使用空字符（Python: '' , Java/C++: '\0' ）做标记是为了防止标记字符与矩阵原有字符重复。当存在重复时，此算法会将矩阵原有字符认作标记字符，从而出现错误。

class Solution {

public:

    bool exist(vector<vector<char>>& board, string word) {

        if(word.empty()){

            return false;

        }

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

            for(int j = 0; j < board[0].size(); ++j){

                if(dfs(board, word, i, j, 0)){

                    return true;

                }

            }

        }

        return false;

    }

    bool dfs(vector<vector<char>>& board, string& word, int i, int j, int w){

        if(i < 0 || i > board.size()-1 || j < 0 || j > board[0].size()-1 || word[w] != board[i][j]){

            return false;

        }

        if(w == word.length() - 1){

            return true;

        }

        char tmp = board[i][j];

        board[i][j] = '\0';

        if(dfs(board, word, i-1, j, w+1)||

           dfs(board, word, i+1, j, w+1)||

           dfs(board, word, i, j-1, w+1)||

           dfs(board, word, i, j+1, w+1)){

               return true;

           }

        board[i][j] = tmp;

        return false;

    }

};

## [剑指 Offer 13. 机器人的运动范围](https://leetcode-cn.com/problems/ji-qi-ren-de-yun-dong-fan-wei-lcof/)

地上有一个m行n列的方格，从坐标 [0,0] 到坐标 [m-1,n-1] 。一个机器人从坐标 [0, 0] 的格子开始移动，它每次可以向左、右、上、下移动一格（不能移动到方格外），也不能进入行坐标和列坐标的数位之和大于k的格子。例如，当k为18时，机器人能够进入方格 [35, 37] ，因为3+5+3+7=18。但它不能进入方格 [35, 38]，因为3+5+3+8=19。请问该机器人能够到达多少个格子？

示例 1：

输入：m = 2, n = 3, k = 1

输出：3

示例 2：

输入：m = 3, n = 1, k = 0

输出：1

提示：

1 <= n,m <= 100

0 <= k <= 20

**解：**

DFS：

时间O(mn) 空间O(mn)

class Solution {

public:

    int res = 0;

    int movingCount(int m, int n, int k) {

        vector<vector<bool>> map(m,vector<bool>(n));

        dfs(map,0,0,m,n,k);

        return res;

    }

    bool jugde(int i, int j, int k){

        int t = 0;

        while(i != 0){

            t += i % 10;

            i /= 10;

        }

        while(j != 0){

            t += j % 10;

            j /= 10;

        }

        if(t > k){

            return true;

        }else

            return false;

    }

    bool dfs(vector<vector<bool>>& map, int i, int j, int m, int n, int k){

        if(i < 0 || i >= m || j < 0 || j >= n || map[i][j] == 1 || jugde(i,j,k))

            return false;

        map[i][j] = 1;

        res++;

        if(dfs(map,i+1,j,m,n,k)||dfs(map,i-1,j,m,n,k)||dfs(map,i,j+1,m,n,k)||dfs(map,i,j-1,m,n,k)){

            return true;

        }

        return false;

    }

};

BFS：

时间O(mn) 空间O(mn)

class Solution {

public:

    int res = 0;

    int movingCount(int m, int n, int k) {

        vector<vector<bool>> visited(m,vector<bool>(n));

        queue<pair<int,int>> q;

        int x = 0;

        int y = 0;

        int res = 0;

        q.emplace(x,y);

        visited[x][y] = true;

        res++;

        while(!q.empty()){

            x = q.front().first;

            y = q.front().second;

            q.pop();

            if(x+1 < m && jugde(x+1,y,k) && visited[x+1][y]==false){

                q.emplace(x+1,y);

                visited[x+1][y] = true;

                res++;

            }

            if(y+1 < n && jugde(x,y+1,k) && visited[x][y+1]==false){

                q.emplace(x,y+1);

                visited[x][y+1] = true;

                res++;

            }

        }

        return res;

    }

    bool jugde(int i, int j, int k){

        int t = 0;

        while(i != 0){

            t += i % 10;

            i /= 10;

        }

        while(j != 0){

            t += j % 10;

            j /= 10;

        }

        if(t > k){

            return false;

        }else

            return true;

    }

};

<https://leetcode-cn.com/problems/ji-qi-ren-de-yun-dong-fan-wei-lcof/solution/mian-shi-ti-13-ji-qi-ren-de-yun-dong-fan-wei-dfs-b/>