**目 录**

[1 二叉树 1](#_Toc98100227)

[1.1 树的子结构 1](#_Toc98100228)

[1.2 二叉树转换为链表 1](#_Toc98100229)

[1.3 从上到下打印二叉树 2](#_Toc98100230)

[1.4 二叉树的序列化 3](#_Toc98100231)

[1.5 最近公共祖先 4](#_Toc98100232)

[2 回溯 5](#_Toc98100233)

[2.1 矩阵中的路径(单词搜索) 5](#_Toc98100234)

[2.2 机器人的运动范围 6](#_Toc98100235)

[2.3 删除无效的括号 7](#_Toc98100236)

[2.4 字符串的排列 9](#_Toc98100237)

[3 动态规划 9](#_Toc98100238)

[3.1 剪绳子 9](#_Toc98100239)

[3.2 最长回文子串 10](#_Toc98100240)

[3.3 正则表达式匹配 11](#_Toc98100241)

[3.4 最佳买卖股票时机含冷冻期 11](#_Toc98100242)

[3.5 丑数 12](#_Toc98100243)

[3.6 n个骰子的点数 13](#_Toc98100244)

[3.7 圆圈中最后剩下的数字（约瑟夫环） 14](#_Toc98100245)

[3.8 分割等和子集（背包问题：二维dp数组） 14](#_Toc98100246)

[3.9 戳气球 15](#_Toc98100247)

[4 数位DP 16](#_Toc98100248)

[4.1 1～n 整数中 1 出现的次数 17](#_Toc98100249)

[5 双指针 18](#_Toc98100250)

[5.1 三数之和 18](#_Toc98100251)

[6 滑动窗口 19](#_Toc98100252)

[6.1 解题框架 19](#_Toc98100253)

[6.2 无重复字符的最长子串 20](#_Toc98100254)

[6.3 和为s的连续正数序列 20](#_Toc98100255)

[7 链表 21](#_Toc98100256)

[7.1 反转链表 21](#_Toc98100257)

[7.2 复杂链表的复制 22](#_Toc98100258)

[7.3 两个链表中的第一个公共节点 22](#_Toc98100259)

[7.4 回文链表 23](#_Toc98100260)

[7.5 寻找重复数 24](#_Toc98100261)

[7.6 找到所有数组中消失的数字 25](#_Toc98100262)

[8 图论 25](#_Toc98100263)

[**8.1** 课程表 25](#_Toc98100264)

[9 基础数据结构 27](#_Toc98100265)

[9.1 堆 27](#_Toc98100266)

[9.2 包含min函数的栈 28](#_Toc98100267)

[9.3 栈的压入、弹出序列 28](#_Toc98100268)

[9.4 最小的k个数 29](#_Toc98100269)

[9.5 数据流中的中位数 30](#_Toc98100270)

[9.6 LRU缓存 31](#_Toc98100271)

[9.7 前K个高频元素 34](#_Toc98100272)

[10 快速幂 35](#_Toc98100273)

[10.1 数值的整数次方 35](#_Toc98100274)

[11 有限状态自动机 35](#_Toc98100275)

[11.1 表示数值的字符串 35](#_Toc98100276)

[12 排序 41](#_Toc98100277)

[12.1 堆排序 41](#_Toc98100278)

[12.2 快排 41](#_Toc98100279)

[12.3 组中的逆序对 42](#_Toc98100280)

[12.4 数组中的第K个最大元素 44](#_Toc98100281)

[13 位运算 44](#_Toc98100282)

[13.1 找出这数组中两个只出现一次的数字 45](#_Toc98100283)

[13.2 求1+2+…+n 45](#_Toc98100284)

[13.3 不用加减乘除做加法 46](#_Toc98100285)

[14 单调队列 46](#_Toc98100286)

[14.1 滑动窗口的最大值 46](#_Toc98100287)

[14.2 队列的最大值 47](#_Toc98100288)

[14.3 柱状图中最大矩形 49](#_Toc98100289)

[14.4 接雨水 50](#_Toc98100290)

[14.5 最大矩形 51](#_Toc98100291)

[14.6 每日温度 52](#_Toc98100292)

[15 前缀树 53](#_Toc98100293)

[16 Union-Find并查集 54](#_Toc98100294)

[16.1 被围绕的区域 55](#_Toc98100295)

[17 前缀和 57](#_Toc98100296)

[18 不知道如何分类（字符串，递归等） 58](#_Toc98100297)

[18.1 字符串解码 58](#_Toc98100298)

1. 二叉树
   1. 树的子结构

输入两棵二叉树A和B，判断B是不是A的子结构。(约定空树不是任意一个树的子结构)

B是A的子结构， 即 A中有出现和B相同的结构和节点值。

例如:

给定的树 A:

     3

    / \

   4   5

  / \

 1   2

给定的树 B：

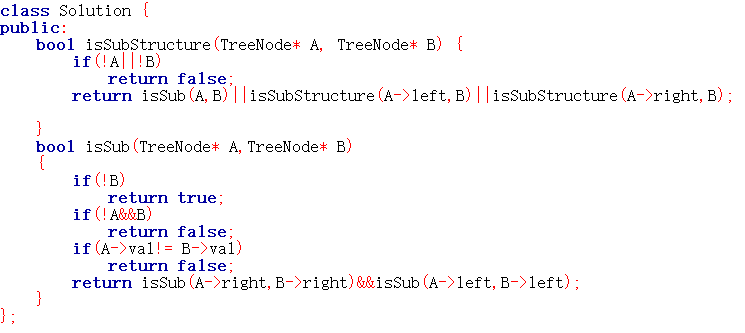
   4

  /

 1

返回 true，因为 B 与 A 的一个子树拥有相同的结构和节点值。

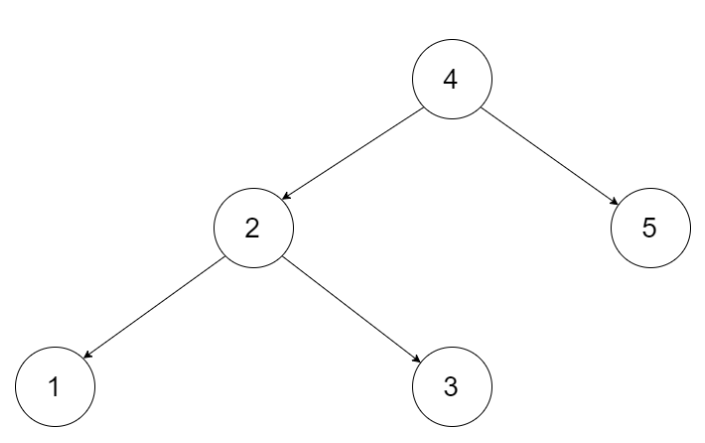
【解题思路】先遍历再递归，先判树A的根节点是否与树B的根节点相同，如果相同在递归判断（前序遍历）树A的子树和树B的子树每个节点是否相等。

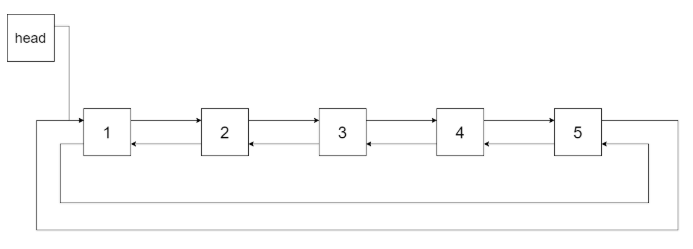


* 1. 二叉树转换为链表

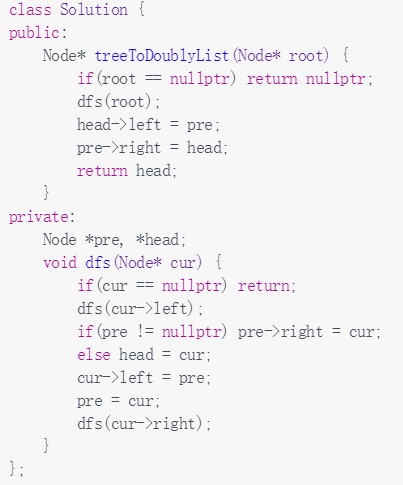
输入一棵二叉搜索树，将该二叉搜索树转换成一个排序的循环双向链表。要求不能创建任何新的节点，只能调整树中节点指针的指向。

为了让您更好地理解问题，以下面的二叉搜索树为例：





【解题思路】中序遍历 改进方法：把pre节点和cur节点储存为全局变量



* 1. 从上到下打印二叉树

请实现一个函数按照之字形顺序打印二叉树，即第一行按照从左到右的顺序打印，第二层按照从右到左的顺序打印，第三行再按照从左到右的顺序打印，其他行以此类推。

【解题思路】常规打印用queue存储，方向转换考虑用deque

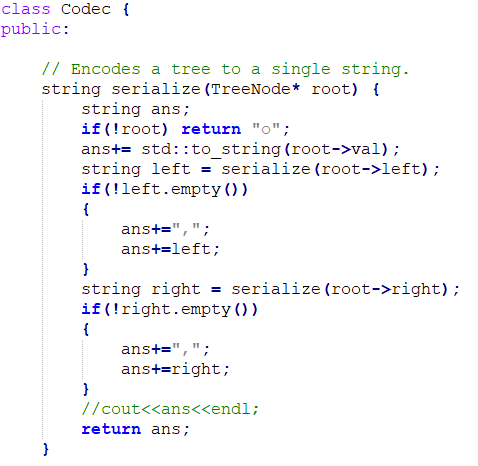


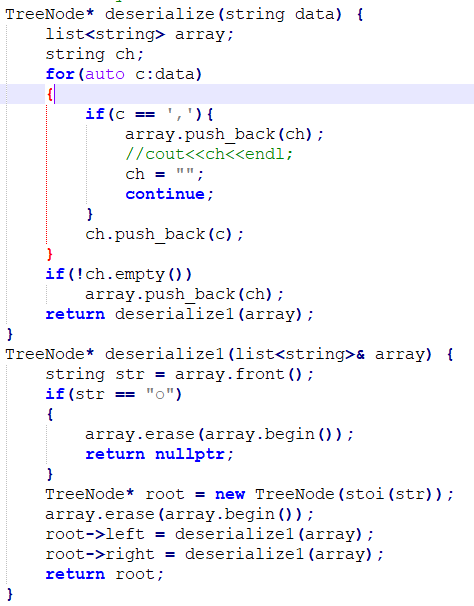
* 1. 二叉树的序列化

请实现两个函数，分别用来序列化和反序列化二叉树。

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

【解题思路】注意序列化时每两个节点之间需要用’,’分隔，反序列化时string随着递归变化，自然的可以遇到递归终点。





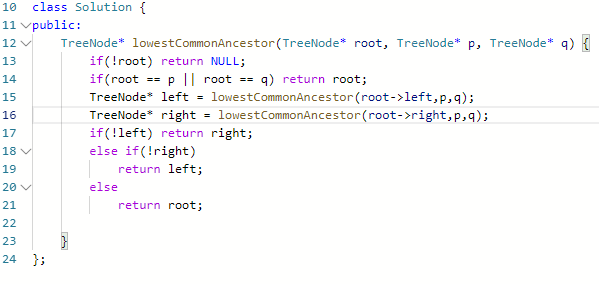
* 1. 最近公共祖先

给定一个二叉树, 找到该树中两个指定节点的最近公共祖先。

百度百科中最近公共祖先的定义为：“对于有根树 T 的两个节点 p、q，最近公共祖先表示为一个节点 x，满足 x 是 p、q 的祖先且 x 的深度尽可能大（一个节点也可以是它自己的祖先）

【解题思路】

1. 如果p,q分别在root的左右子树中，即（左右子树的递归结果都不为0），则返回root；
2. 如果p,q都在root的左子树中（左子树递归结果不为0，右子树递归结果为0），则func(root->left, p, q)是结果；
3. 如果p,q都在root的右子树中（左子树递归结果为0，右子树递归结果为0），则func(root->right,p,q)是结果。



* 1. 求根节点到叶节点数字之和

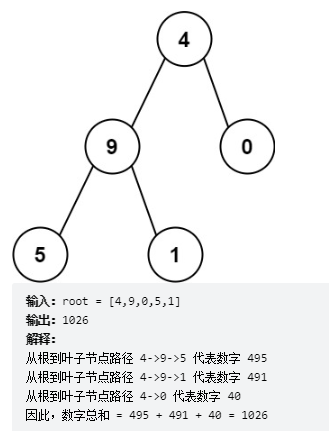
给你一个二叉树的根节点 root ，树中每个节点都存放有一个 0 到 9 之间的数字。

每条从根节点到叶节点的路径都代表一个数字：

例如，从根节点到叶节点的路径 1 -> 2 -> 3 表示数字 123 。

计算从根节点到叶节点生成的 所有数字之和 。

叶节点 是指没有子节点的节点。



class Solution {

public:

    int sumNumbers(TreeNode\* root) {

        return sumNumbersHelp(root,0);

    }

    int sumNumbersHelp(TreeNode\* root, int preSum){

        if(!root)

            return 0;

        if(!root->left&&!root->right)

        {

            return root->val + preSum\*10;

        }

        preSum = preSum\*10 + root->val;

        int leftVal = sumNumbersHelp(root->left,preSum);

        int rightVal = sumNumbersHelp(root->right,preSum);

        return leftVal+rightVal;

    }

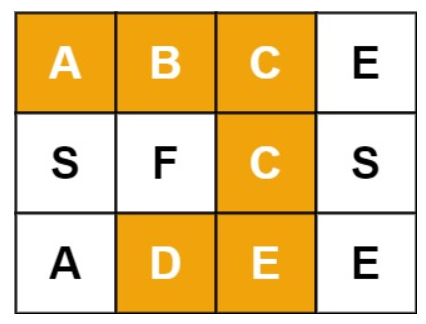
};

1. 回溯
   1. 矩阵中的路径(单词搜索)

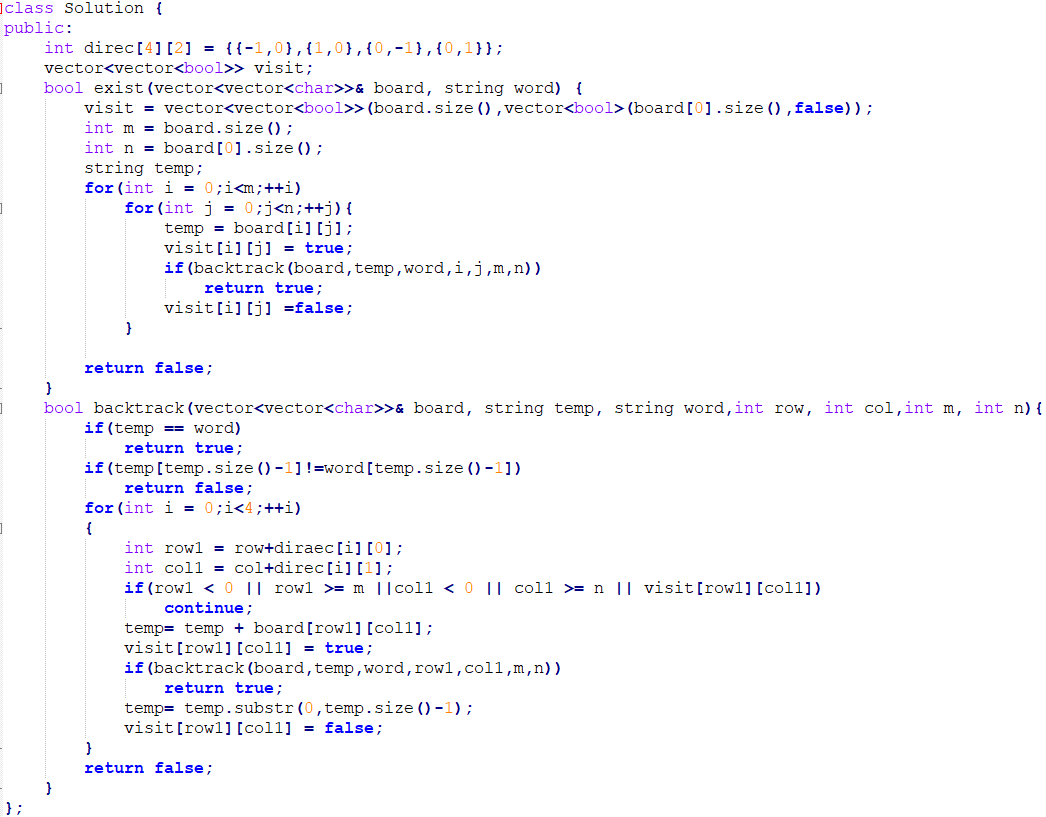
给定一个 m x n 二维字符网格 board 和一个字符串单词 word 。如果 word 存在于网格中，返回 true ；否则，返回 false 。

单词必须按照字母顺序，通过相邻的单元格内的字母构成，其中“相邻”单元格是那些水平相邻或垂直相邻的单元格。同一个单元格内的字母不允许被重复使用。

例如，在下面的 3×4 的矩阵中包含单词 "ABCCED"（单词中的字母已标出）



**【解题思路】**非常明显的回溯题，注意：1. 在主函数里面需要遍历二维数组获取单词起点，再进行回溯；2. 根据每一位字符进行剪枝，而不是每次判断整个单词是否相等。



* 1. 机器人的运动范围

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？

**【解题思路】**回溯时需注意两个问题，1，递归不回溯，返回后visit不需要设回false；2，只需要向右向下移动即可，不需要遍历四个方向。



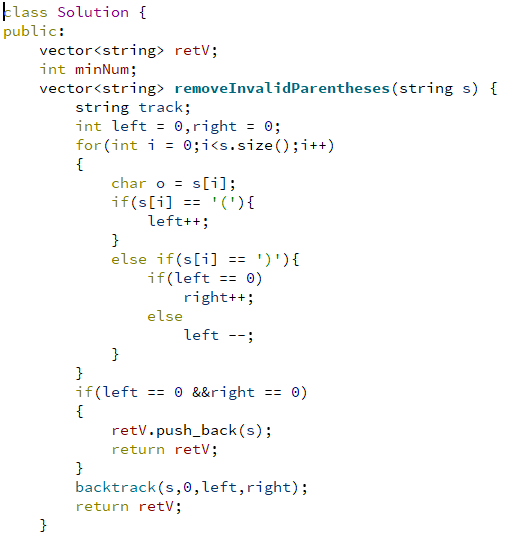
* 1. 删除无效的括号

给你一个由若干括号和字母组成的字符串 s ，删除最小数量的无效括号，使得输入的字符串有效。

返回所有可能的结果。答案可以按 任意顺序 返回。



【解题思路】首先，根据s中左右括号的个数，计算多出来的左或右括号个数。第二，回溯。第三，回溯过程中需要写一个辅助函数isValid用来判断字符串是否有效。





* 1. 字符串的排列

输入一个字符串，打印出该字符串中字符的所有排列。

你可以以任意顺序返回这个字符串数组，但里面不能有重复元素。

【解题思路】关键在于剪枝和去重，见下方红色代码

class Solution {

public:

    vector<string> ans;

    vector<bool> visit;

    vector<string> permutation(string s) {

        visit = vector<bool>(s.size(),false);

        sort(s.begin(),s.end());

        string temp = "";

        backtrack(s,temp);

        return ans;

    }

    void backtrack(string s, string temp)

    {

        if(temp.size() == s.size())

            ans.push\_back(temp);

        for(int i = 0;i<s.size();i++)

        {

            if(visit[i])

                continue;

            if(i>0&&s[i] == s[i-1]&&!visit[i-1])

                continue;

            temp.push\_back(s[i]);

            visit[i] = true;

            backtrack(s,temp);

            temp.pop\_back();

            visit[i] = false;

        }

    }

};

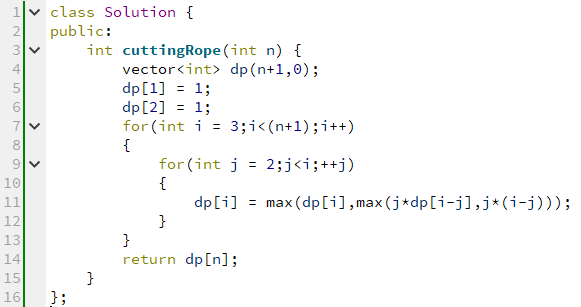
1. 动态规划
   1. 剪绳子

给你一根长度为 n 的绳子，请把绳子剪成整数长度的 m 段（m、n都是整数，n>1并且m>1），每段绳子的长度记为 k[0],k[1]...k[m-1] 。请问 k[0]\*k[1]\*...\*k[m-1] 可能的最大乘积是多少？例如，当绳子的长度是8时，我们把它剪成长度分别为2、3、3的三段，此时得到的最大乘积是18。

**【解题思路】**能想到是动态规划，取最大值时实际应该是：（共四项）因为m>1，其中(i-j)\*j表示只剪一刀，dp[i-j]\*dp[j]表示至少3刀，dp[i-j]\*j,和dp[j]\*[i-j]表示至少两刀。

max(dp[i-j]\*dp[j], (i-j)\*j, dp[i-j]\*j, dp[j]\*[i-j])

因为会进行迭代，所以有些像将是重复的，可以只保留其中两项。max(j\*dp[i-j],j\*(i-j))



* 1. 最长回文子串

给你一个字符串 s，找到 s 中最长的回文子串。

【解题思路】字符串子串的动态规划问题，需要定义2维dp数组，i,j分别表示子串的首位index。



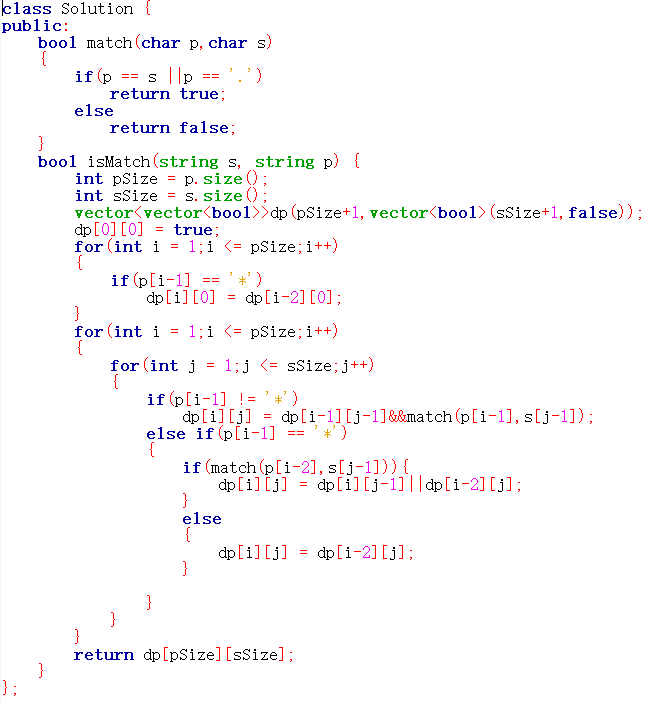
* 1. 正则表达式匹配

请实现一个函数用来匹配包含'. '和'\*'的正则表达式。模式中的字符'.'表示任意一个字符，而'\*'表示它前面的字符可以出现任意次（含0次）。在本题中，匹配是指字符串的所有字符匹配整个模式。例如，字符串"aaa"与模式"a.a"和"ab\*ac\*a"匹配，但与"aa.a"和"ab\*a"均不匹配。

【解题思路】难点在于遇到“\*”时状态转移表达式的写法，分两种情况考虑。

1. p[i-1] == s[j]，dp[i][j] = dp[i][j-1]||dp[i-2][j]
2. p[i-1] != s[j]，dp[i][j] = dp[i-2][j]

设计dp数组时，应使dp数组长宽比s,p长度大1，初始状态dp[0][0]为true，且x\*y\*这种类型的正则表达式可以和空字符串匹配。



* 1. 最佳买卖股票时机含冷冻期

给定一个整数数组prices，其中第  prices[i] 表示第 i 天的股票价格 。​

设计一个算法计算出最大利润。在满足以下约束条件下，你可以尽可能地完成更多的交易（多次买卖一支股票）:

卖出股票后，你无法在第二天买入股票 (即冷冻期为 1 天)。

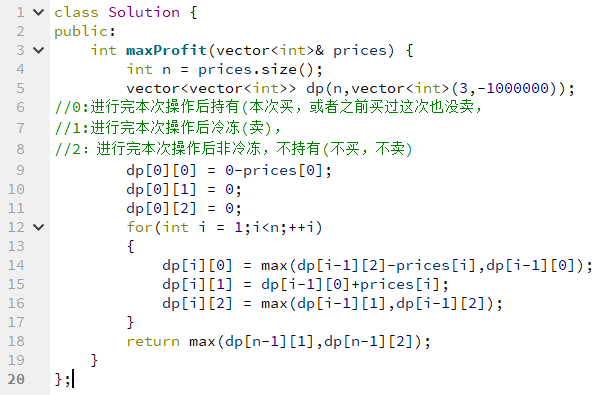
注意：你不能同时参与多笔交易（你必须在再次购买前出售掉之前的股票）。

【解题思路】二维dp数组定义vector<vector<int>> dp(n,vector<int>(3,-1000000));

//0:进行完本次操作后持有(本次买，或者之前买过这次也没卖，

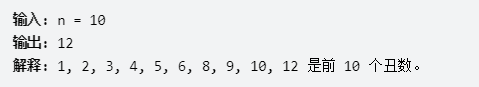
//1:进行完本次操作后冷冻(卖)，

//2：进行完本次操作后非冷冻，不持有(不买，不卖)

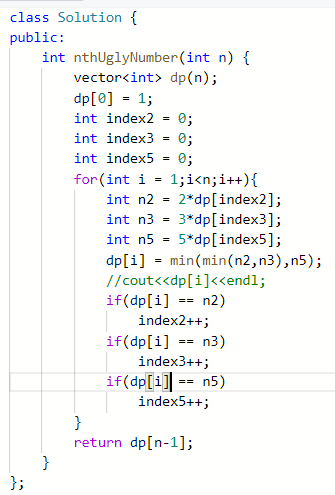


* 1. 丑数

我们把只包含质因子 2、3 和 5 的数称作丑数（Ugly Number）。求按从小到大的顺序的第 n 个丑数。



【解题思路】后面的丑数是由前方的丑数\*2/\*3/\*5得来的，用三个index记录\*2/\*3/\*5的数据位置，每次取三个中最小的数，并把该index++。

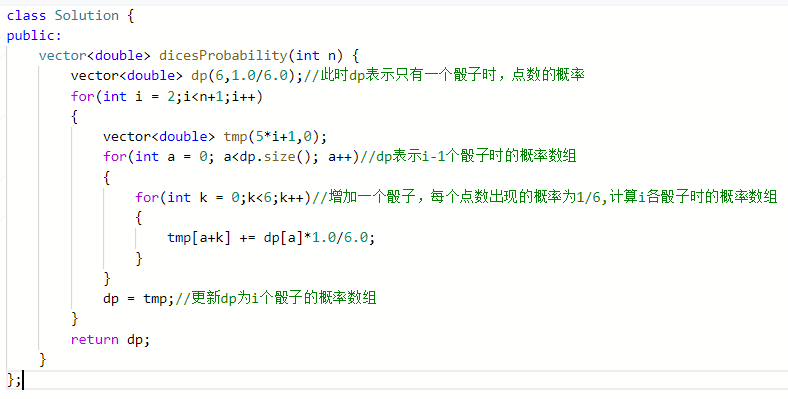


* 1. n个骰子的点数

把n个骰子扔在地上，所有骰子朝上一面的点数之和为s。输入n，打印出s的所有可能的值出现的概率。

你需要用一个浮点数数组返回答案，其中第 i 个元素代表这 n 个骰子所能掷出的点数集合中第 i 小的那个的概率。

【解题思路】n= 1时，每个点数的概率都为1/6，n每次递增1，每个点数的概率可以由n-1的概率计算得出，因此每次n++后都要更新dp数组。



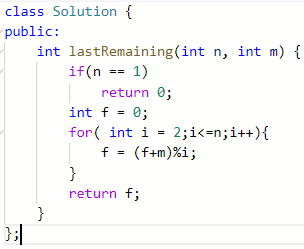
* 1. 圆圈中最后剩下的数字（约瑟夫环）

0,1,···,n-1这n个数字排成一个圆圈，从数字0开始，每次从这个圆圈里删除第m个数字（删除后从下一个数字开始计数）。求出这个圆圈里剩下的最后一个数字。

例如，0、1、2、3、4这5个数字组成一个圆圈，从数字0开始每次删除第3个数字，则删除的前4个数字依次是2、0、4、1，因此最后剩下的数字是3。

【解题思路】首先，长度为 n 的序列会先删除第 m % n 个元素，然后剩下一个长度为 n - 1 的序列。那么，我们可以递归地求解 f(n - 1, m)，就可以知道对于剩下的 n - 1 个元素，最终会留下第几个元素，我们设答案为 x = f(n - 1, m)。

由于我们删除了第 m % n 个元素，将序列的长度变为 n - 1。当我们知道了 f(n - 1, m) 对应的答案 x 之后，我们也就可以知道，长度为 n 的序列最后一个删除的元素，应当是从 m % n 开始数的第 x 个元素。因此有 f(n, m) = (m % n + x) % n = (m + x) % n。

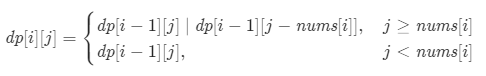


* 1. 分割等和子集（背包问题：二维dp数组）

给你一个 只包含正整数 的 非空 数组 nums 。请你判断是否可以将这个数组分割成两个子集，使得两个子集的元素和相等。

【解题思路】先遍历数组累加，获取到所有元素的和，如果和为奇数，就返回false，如果是偶数就获取到target=sum/2，转换为背包问题。

定义二维dp数组dp[i][j]，i表示数组中的第i个元素，j表示需要凑成的数字和。可以得到转移方程：



    bool canPartition(vector<int>& nums) {

        //动态规划，转化为01背包问题

        int sum = 0;

        for(auto & n:nums){

            sum += n;

        }

        if(sum%2 != 0)

            return false;

        int target = sum/2;

        vector<vector<bool>>dp(nums.size()+1,vector<bool>(target+1,false));

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

            dp[i][0] = true;

        }

        dp[0][0] = true;

        for(int t = 0;t<=target;t++){

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

                dp[i][t] = dp[i-1][t]||(t-nums[i-1]>=0?dp[i-1][t-nums[i-1]]:false);

            }

        }

        return dp[nums.size()][target];

    }

* 1. 戳气球

有 n 个气球，编号为0 到 n - 1，每个气球上都标有一个数字，这些数字存在数组 nums 中。

现在要求你戳破所有的气球。戳破第 i 个气球，你可以获得 nums[i - 1] \* nums[i] \* nums[i + 1] 枚硬币。 这里的 i - 1 和 i + 1 代表和 i 相邻的两个气球的序号。如果 i - 1或 i + 1 超出了数组的边界，那么就当它是一个数字为 1 的气球。

求所能获得硬币的最大数量。

。

【解题思路】对于一个开区间(i,j)，如果k是这个区间最后一个被戳破的气球，dp[i][j]表示开区间(i,j)内能拿到的最多金币，total = dp[i][k]+val[i]\*val[k]\*val[j]+dp[k][j]，dp[i][j]等于遍历i~j区间后得到的最大total。

class Solution {

public:

    int maxCoins(vector<int>& nums) {

        int n = nums.size();

        vector<vector<int>>dp(n+2,vector<int>(n+2,0));

        vector<int> val(n+2);

        val[0] = 1;

        val[n+1] = 1;

        for(int i = 0;i<n;i++){

            val[i+1] = nums[i];

        }

        for(int i = n-1;i>=0;i--){

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

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

                    int sum = val[k]\*val[i]\*val[j];

                    sum += dp[i][k]+dp[k][j];

                    dp[i][j] = max(dp[i][j],sum);

                }

            }

        }

        return dp[0][n+1];

    }

};

* 1. 编辑距离

给你两个单词 word1 和 word2， 请返回将 word1 转换成 word2 所使用的最少操作数。

你可以对一个单词进行如下三种操作：

插入一个字符

删除一个字符

替换一个字符

class Solution {

public:

    int minDistance(string word1, string word2) {

        vector<vector<int>> dp(word1.size()+1,vector<int>(word2.size()+1,0));

        for(int j = 1;j<=word2.size();j++){

            dp[0][j] = j;

        }

        for(int i = 1;i<=word1.size();i++){

            dp[i][0] = i;

        }

        for(int i = 1;i<=word1.size();i++){

            for(int j = 1;j<=word2.size();j++){

                if(word1[i-1] == word2[j-1])

                {

                    dp[i][j] = dp[i-1][j-1];

                }else{

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

                }

            }

        }

        return dp[word1.size()][word2.size()];

    }

};

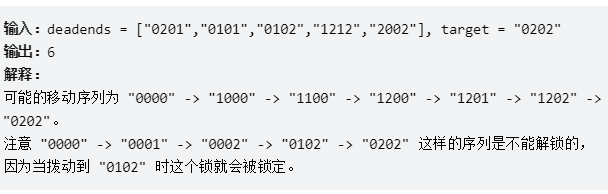
1. 广度优先搜索（BFS）
   1. 打开转盘锁

你有一个带有四个圆形拨轮的转盘锁。每个拨轮都有10个数字： '0', '1', '2', '3', '4', '5', '6', '7', '8', '9' 。每个拨轮可以自由旋转：例如把 '9' 变为 '0'，'0' 变为 '9' 。每次旋转都只能旋转一个拨轮的一位数字。

锁的初始数字为 '0000' ，一个代表四个拨轮的数字的字符串。

列表 deadends 包含了一组死亡数字，一旦拨轮的数字和列表里的任何一个元素相同，这个锁将会被永久锁定，无法再被旋转。

字符串 target 代表可以解锁的数字，你需要给出解锁需要的最小旋转次数，如果无论如何不能解锁，返回 -1 。



class Solution {

public:

    int openLock(vector<string>& deadends, string target) {

        if(target == "0000") return 0;

        queue<string> que;

        unordered\_set<string> deadset(deadends.begin(),deadends.end());

        if(deadset.count("0000")) return -1;

        unordered\_map<string,int> visit;

        auto numnext = [](char x) {return x=='9'?'0':x+1;};

        auto numpre = [](char x) {return x=='0'?'9':x-1;};

        auto get = [&](string &status) {

            vector<string> ret;

            for(int i = 0;i < 4;i++){

                char n = status[i];

                status[i] = numnext(n);

                ret.push\_back(status);

                status[i] = numpre(n);

                ret.push\_back(status);

                status[i] = n;

            }

            return ret;

        };

        que.push("0000");

        visit["0000"] = 0;

        int step = 0;

        while(!que.empty()){

            string ss = que.front();

            if(visit.count(ss))

                step = visit[ss];

            que.pop();

            for(auto&& s:get(ss)){

                if(visit.count(s)||deadset.count(s))

                    continue;

                else{

                    if(target == s) return step+1;

                    visit[s] = step+1;

                    que.push(s);

                }

            };

        }

        return -1;

    }

};

1. 数位DP

**数位dp解决的问题：**

求出在给定区间[A,B][A,B]内，符合条件f(i)f(i)的数ii的个数。条件f(i)f(i)一般与数的大小无关，而与数的组成有关。

**Dp数组的定义**：dp[pos][pre][st]… pos表示数组的第几位（从高到低），pre表示前几位数，st表示答案。

**Dfs函数参量：**pos表示第几位，limit表示高位是否被限制，lead表示前导0。

**前导0（lead）**：加入要寻找任意相邻两数相等的数，111，222，888是符合题意的数，但是如果考虑三位数由（0111，0222，0888）表示就不符合题意了。所以需要加一个前导0标记。

1. 如果当前位lead=1lead=1而且当前位也是0，那么当前位也是前导0，pos+1继续搜；
2. 如果当前位lead=1lead=1但当前位不是0，则本位作为当前数的最高位，pos+1继续搜；

有时前导0不需要判断。

**最高位标记：**

1. 若当前位limit=1而且已经取到了能取到的最高位时，下一位limit=1；
2. 若当前位limit=1但是没有取到能取到的最高位时，下一位limit=0；
3. 若当前位limit=0时，下一位limit=0。

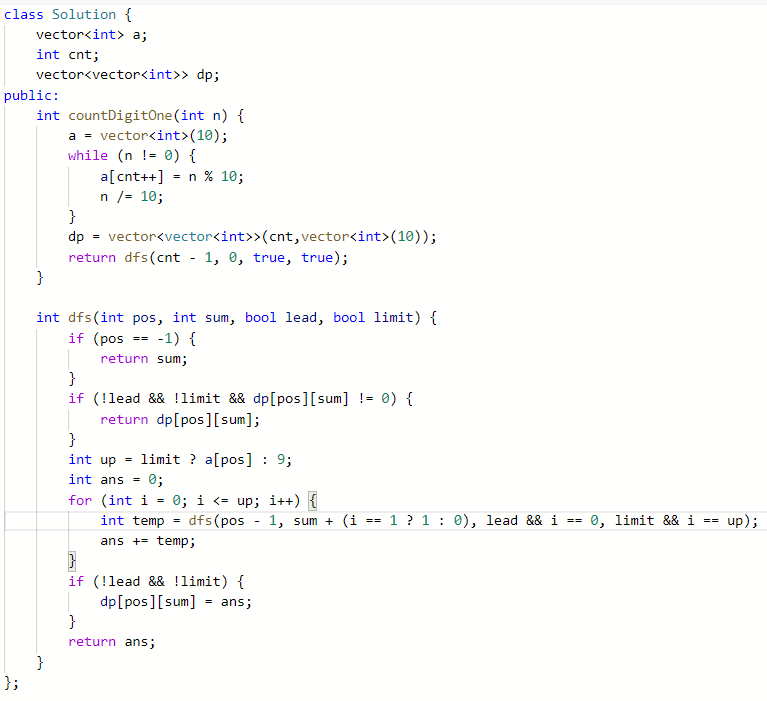
**模板：**



* 1. 1～n 整数中 1 出现的次数

输入一个整数 n ，求1～n这n个整数的十进制表示中1出现的次数。

例如，输入12，1～12这些整数中包含1 的数字有1、10、11和12，1一共出现了5次。



1. 双指针
   1. 三数之和

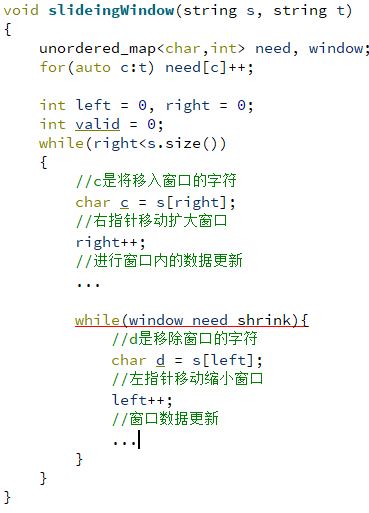
给你一个包含 n 个整数的数组 nums，判断 nums 中是否存在三个元素 a，b，c ，使得 a + b + c = 0 ？请你找出所有和为 0 且不重复的三元组。

注意：答案中不可以包含重复的三元组。

【解题思路】首先排序（排序后才能使用双指针？），遍历数组取第一个数，第二个数和第三个数采用双指针，题目的重点在于去重。



1. 滑动窗口
   1. 解题框架



* 1. 无重复字符的最长子串

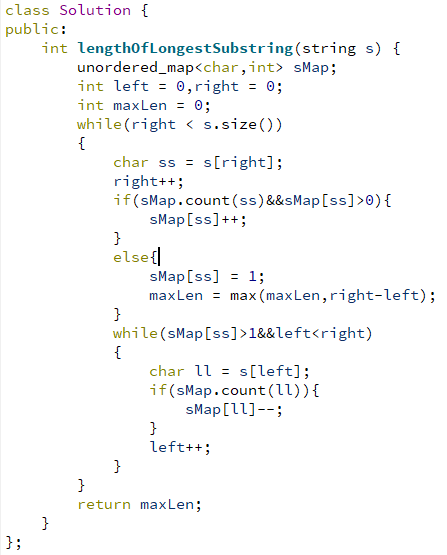
给定一个字符串 s ，请你找出其中不含有重复字符的 **最长子串**的长度。

**输入:** s = "abcabcbb"

**输出:** 3

**解释:** 因为无重复字符的最长子串是 "abc"，所以其长度为 3。

【解题思路】双指针，套模板



* 1. 和为s的连续正数序列

输入一个正整数 target ，输出所有和为 target 的连续正整数序列（至少含有两个数）。

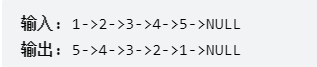
序列内的数字由小到大排列，不同序列按照首个数字从小到大排列。

【解题思路】滑动窗口

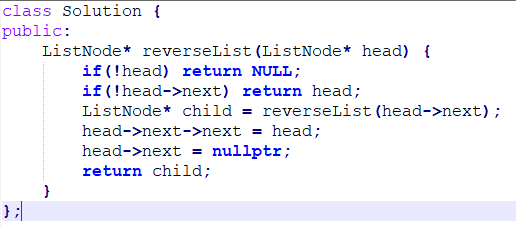


1. 链表
   1. 反转链表

定义一个函数，输入一个链表的头节点，反转该链表并输出反转后链表的头节点。

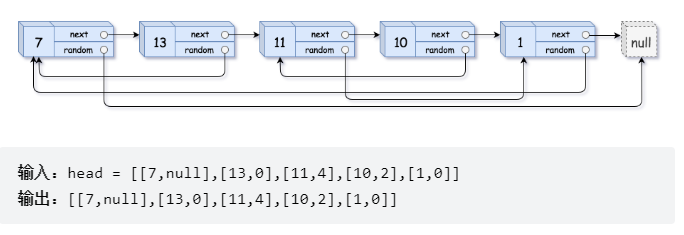


**【解题思路】**两种方法：递归、迭代。注意答案的最后一个节点需要指向null。

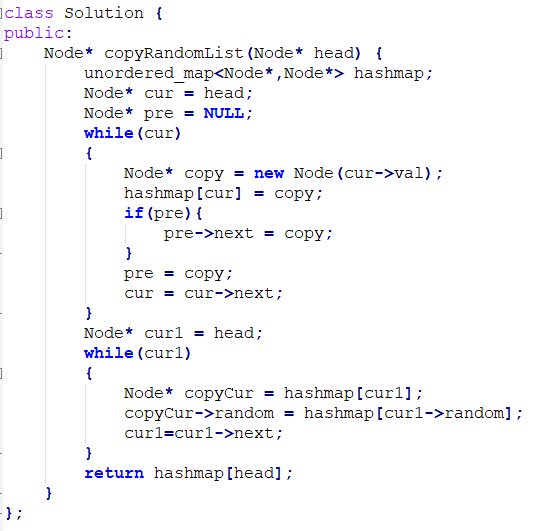


* 1. 复杂链表的复制

请实现 copyRandomList 函数，复制一个复杂链表。在复杂链表中，每个节点除了有一个 next 指针指向下一个节点，还有一个 random 指针指向链表中的任意节点或者 null。



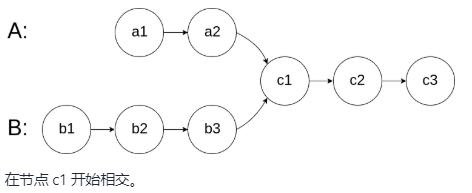
【解题思路】利用hashmap存储新旧链表节点的对应关系，第一遍遍历先把next连好，第二遍遍历在连random



* 1. 两个链表中的第一个公共节点

输入两个链表，找出它们的第一个公共节点。

如下面的两个链表：



【解题思路】两个链表连接起来形成两个新链表，第一个是A在前，B在后；第二个是B在前，A在后，然后同步遍历，遇到的第一个相同节点就是第一个公共节点。

* 1. 回文链表

给你一个单链表的头节点 head ，请你判断该链表是否为回文链表。如果是，返回 true ；否则，返回 false 。

【解题思路】首先利用快慢指针获取到链表一半的位置，把后一半链表反转，如果是回文链表，应该与前一半链表相同。

class Solution {

public:

    bool isPalindrome(ListNode\* head) {

        ListNode\* slow = head, \*fast = head;

        while(fast&&fast->next){

            slow = slow->next;

            fast = fast->next->next;

        }

        ListNode\* head2;

        if(fast){//奇数

            head2 = reverse(slow->next);

        }else

        {

            head2 = reverse(slow);

        }

        ListNode\* head1 = head;

        while(head1&&head2){

            if(head1->val!=head2->val)

                return false;

            head1 = head1->next;

            head2 = head2->next;

        }

        return true;

    }

    ListNode\* reverse(ListNode\* head){

        ListNode\* pre = nullptr;

        ListNode\* cur = head;

        while(cur){

            ListNode\* next = cur->next;

            cur->next = pre;

            pre = cur;

            cur = next;

        }

        return pre;

    }

};

* 1. 寻找重复数

给定一个包含 n + 1 个整数的数组 nums ，其数字都在 [1, n] 范围内（包括 1 和 n），可知至少存在一个重复的整数。

假设 nums 只有 一个重复的整数 ，返回 这个重复的数 。

你设计的解决方案必须 不修改 数组 nums 且只用常量级 O(1) 的额外空间。

【解题思路】

 这种**所有数字取值范围在[1,n]内的数组**可以被当作链表来考虑，数值可以被当作index获取到下一个数字。

class Solution {

public:

    int findDuplicate(vector<int>& nums) {

        //快慢指针

        //这道题的关键在于数组的值可以当index用

        //有重复数相当于是一个有环的链表

        int left = 0,right = 0;

        do{

            left = nums[left];

            right = nums[nums[right]];

        }while(left!=right);

        left = 0;

        while(left!=right){

            left = nums[left];

            right = nums[right];

        }

        return left;

    }

};

* 1. 找到所有数组中消失的数字

给你一个含 n 个整数的数组 nums ，其中 nums[i] 在区间 [1, n] 内。请你找出所有在 [1, n] 范围内但没有出现在 nums 中的数字，并以数组的形式返回结果。

你能在不使用额外空间且时间复杂度为O(n)的情况下解决这个问题吗? 你可以假定返回的数组不算在额外空间内。

【解题思路】这种**所有数字取值范围在[1,n]内的数组**可以被当作链表来考虑，数值可以被当作index获取到下一个数字。遍历数组把能取到的地方的值都变为负数，下次遍历时仍为正数的index就是缺失的数字。

class Solution {

public:

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

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

            nums[abs(nums[i])-1] = -abs(nums[abs(nums[i])-1]);

        }

        vector<int> ans;

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

            if(nums[i] > 0){

                ans.push\_back(i+1);

            }

        }

        return ans;

    }

};

* 1. K个一组翻转链表

给你一个链表，每 k 个节点一组进行翻转，请你返回翻转后的链表。

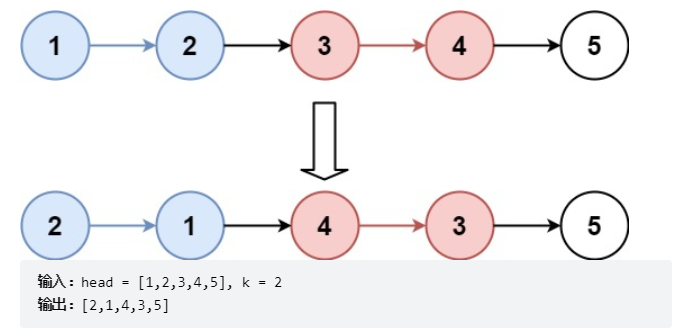
k 是一个正整数，它的值小于或等于链表的长度。

如果节点总数不是 k 的整数倍，那么请将最后剩余的节点保持原有顺序。

进阶：

你可以设计一个只使用常数额外空间的算法来解决此问题吗？

你不能只是单纯的改变节点内部的值，而是需要实际进行节点交换。



【解题思路】先写一个通用的reverse函数，然后遍历链表，每K个调用一下reverse来反转，定义一个前置节点dummy会方便很多。

class Solution {

public:

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

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

    dummy->next = head;

    ListNode\* pre = dummy;

    ListNode\* end = dummy;

    while(end->next != nullptr){

        for(int i = 0;i<k&&end!=nullptr;i++) end = end->next;

        if(end == nullptr) break;

        ListNode\* start = pre->next;

        ListNode\* next = end->next;

        end->next = nullptr;

        pre->next = reverse(start);

        start->next = next;

        pre = start;

        end = pre;

    }

    return dummy->next;

}

ListNode\* reverse(ListNode\* head){

    ListNode\* preNode = nullptr;

    ListNode\* cur = head;

    while(cur!=nullptr){

        ListNode\* next = cur->next;

        cur->next = preNode;

        preNode = cur;

        cur = next;

    }

    return preNode;

}

}

* 1. 链表排序

给你链表的头结点 head ，请将其按 **升序** 排列并返回 **排序后的链表** 。

class Solution {

public:

    ListNode\* sortList(ListNode\* head) {

        //递归终点

        if(!head) return nullptr;

        if(!head->next) return head;

        ListNode\* slow = head;

        ListNode\* fast = head;

        //快慢指针对链表二分

        while (fast->next!=nullptr&&fast->next->next!=nullptr){

            slow = slow->next;

            fast = fast->next->next;

        }

        ListNode\* head2 = slow->next;

        slow->next = nullptr;

        ListNode\* front = sortList(head);

        ListNode\* back = sortList(head2);

        //前导头节点

        ListNode\* preHeadNew = new ListNode(0);

        ListNode\* cur = preHeadNew;

        //归并排序

        while(front&&back){

            if(front->val<back->val){

                cur->next = front;

                front = front->next;

                cur = cur->next;

            } else

            {

                cur->next = back;

                back = back->next;

                cur = cur->next;

            }

        }

        if(front) cur->next = front;

        if(back) cur->next = back;

        return preHeadNew->next;

    }

};

1. 图论
   1. 课程表

你这个学期必须选修 numCourses 门课程，记为 0 到 numCourses - 1 。

在选修某些课程之前需要一些先修课程。 先修课程按数组 prerequisites 给出，其中 prerequisites[i] = [ai, bi] ，表示如果要学习课程 ai 则 必须 先学习课程  bi 。

例如，先修课程对 [0, 1] 表示：想要学习课程 0 ，你需要先完成课程 1 。

请你判断是否可能完成所有课程的学习？如果可以，返回 true ；否则，返回 false 。

【解题思路】1.先构建邻接表，再通过深度优先搜索判断图是否有环。

1. 拓扑排序，先构建邻接表和入度表，建立一个queue用来临时储存入度为0的节点。

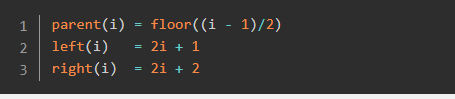
入度为0表示该课程没有任何依赖，可以进行学习，每次从queue中取出一个入度为0的节点，并把以该节点为依赖的所有节点的入度减1，如果入度减为0就push进queue，最后查看能够学习的课程数目是否等于课程总数。





1. 基础数据结构
   1. 堆

近似于一个完全二叉树，一般用数组来表示。如果 i 是节点的索引，那么下面的公式就给出了它的父节点和子节点在数组中的位置：



堆的实现，关键点在于插入和移除时数组的重构。

插入（上浮）时，先插入到数组尾部，然后向上查找，如果大于父节点，就进行交换。直到其值小于父节点为止。

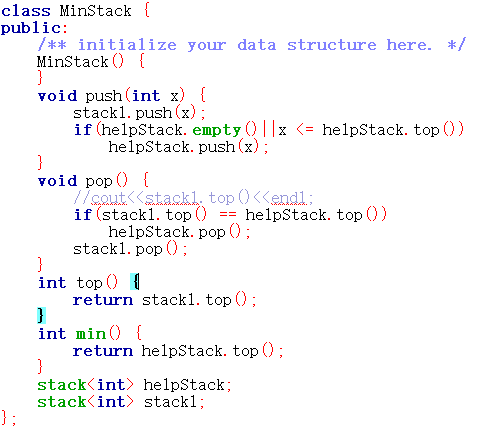
取出堆顶元素（下沉）时，先将数组的最后一个元素替换堆顶元素，删除最后一个元素，然后将其分别与左右孩子子节点比较，选出较大的孩子节点和其进行交换。知道其变为叶子节点为止（没有孩子节点）

上浮和下沉的时间复杂度都为O(logn)

* 1. 包含min函数的栈

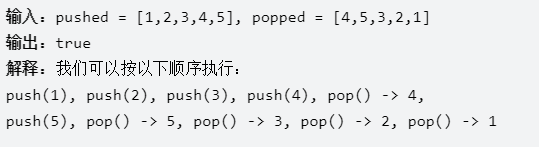
定义栈的数据结构，请在该类型中实现一个能够得到栈的最小元素的 min 函数在该栈中，调用 min、push 及 pop 的时间复杂度都是 O(1)。

【解题思路】定义辅助栈，栈顶永远存储min

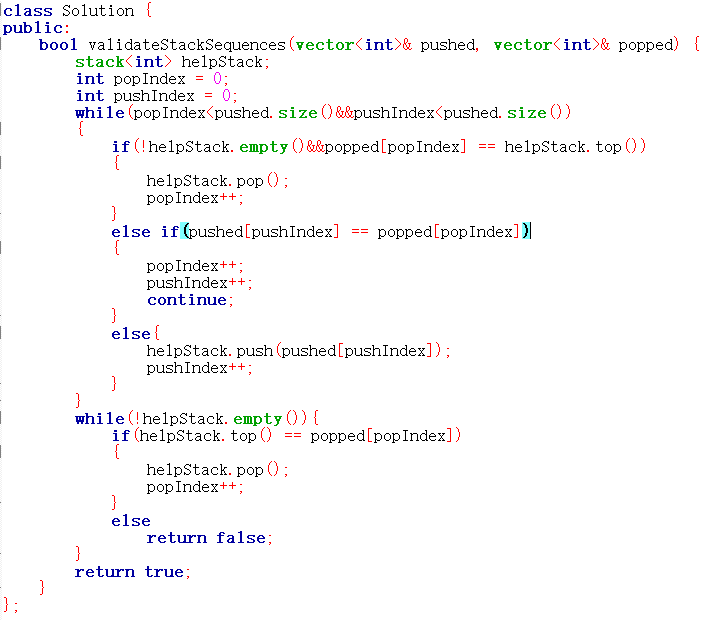


* 1. 栈的压入、弹出序列

输入两个整数序列，第一个序列表示栈的压入顺序，请判断第二个序列是否为该栈的弹出顺序。假设压入栈的所有数字均不相等。例如，序列 {1,2,3,4,5} 是某栈的压栈序列，序列 {4,5,3,2,1} 是该压栈序列对应的一个弹出序列，但 {4,3,5,1,2} 就不可能是该压栈序列的弹出序列。



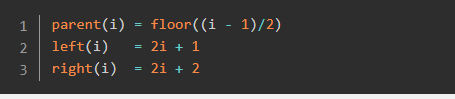
【解题思路】定义辅助栈

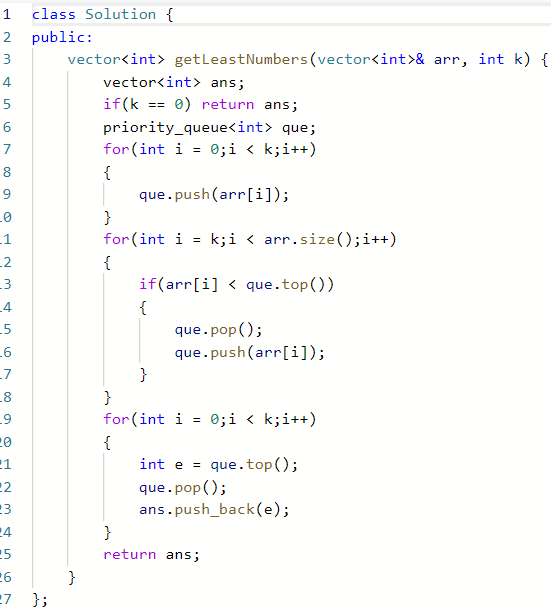


* 1. 最小的k个数

输入整数数组 arr ，找出其中最小的 k 个数。例如，输入4、5、1、6、2、7、3、8这8个数字，则最小的4个数字是1、2、3、4。

【解题思路】利用priority\_queue来实现，priority\_queue底层是由大根堆实现的，默认内部的数据由大到小排列。大根堆近似于完全二叉树，可以由一维数组来表示：[ 10, 7, 2, 5, 1 ]，其根节点数值比两个子节点大，如果 i 是节点的索引，那么下面的公式就给出了它的父节点和子节点在数组中的位置：





* 1. 数据流中的中位数

如何得到一个数据流中的中位数？如果从数据流中读出奇数个数值，那么中位数就是所有数值排序之后位于中间的数值。如果从数据流中读出偶数个数值，那么中位数就是所有数值排序之后中间两个数的平均值。

例如，

[2,3,4] 的中位数是 3

[2,3] 的中位数是 (2 + 3) / 2 = 2.5

设计一个支持以下两种操作的数据结构：

void addNum(int num) - 从数据流中添加一个整数到数据结构中。

double findMedian() - 返回目前所有元素的中位数。

【解题思路】利用二叉堆的特性，创建一个大顶堆和一个小顶堆，大顶堆存储较小的一半数据，小顶堆存储较大的一半数据。



* 1. LRU缓存

请你设计并实现一个满足  LRU (最近最少使用) 缓存 约束的数据结构。

实现 LRUCache 类：

LRUCache(int capacity) 以 正整数 作为容量 capacity 初始化 LRU 缓存

int get(int key) 如果关键字 key 存在于缓存中，则返回关键字的值，否则返回 -1 。

void put(int key, int value) 如果关键字 key 已经存在，则变更其数据值 value ；如果不存在，则向缓存中插入该组 key-value 。如果插入操作导致关键字数量超过 capacity ，则应该 逐出 最久未使用的关键字。

函数 get 和 put 必须以 O(1) 的平均时间复杂度运行

【解题思路】Java中的LinkedHashMap，用哈希表和双向链表来实现，每次取一个数据时，都把这个数据移动到链表头部。

class LRUCache {

    struct biDirecList{

        biDirecList\* pre;

        biDirecList\* next;

        int key, value;

        biDirecList():key(0),value(0),pre(nullptr),next(nullptr){}

        biDirecList(int \_key, int \_value):key(\_key),value(\_value),pre(nullptr),next(nullptr){}

    };

public:

    biDirecList\* head;

    biDirecList\* tail;

    unordered\_map<int,biDirecList\*> cache;

    int \_len;

    int  \_capacity;

    LRUCache(int capacity) {

        head = new biDirecList();

        tail = new biDirecList();

        head->next = tail;

        tail->pre = head;

        \_len = 0;

        \_capacity = capacity;

    }

    int get(int key) {

        if(!cache.count(key)) {

            return -1;

        } else {

            biDirecList\* node = cache[key];

            moveToHead(node);

            return node->value;

        }

    }

    void put(int key, int value) {

        if(cache.count(key))

        {

            cache[key]->value = value;

            moveToHead(cache[key]);

        } else{

            if(++\_len>\_capacity){

                biDirecList\* node = new biDirecList(key,value);

                biDirecList\* nodeEnd = tail->pre;

                cache.erase(nodeEnd->key);

                removeTail();

                delete nodeEnd;

                addHead(node);

                cache[node->key] = node;

            }else{

                biDirecList\* node = new biDirecList(key,value);

                addHead(node);

                cache[node->key] = node;

            }

        }

    }

    void addHead(biDirecList\* node){

        biDirecList\* tmp = head->next;

        head->next = node;

        node->next = tmp;

        tmp->pre = node;

        node->pre = head;

    }

    void removeTail(){

        biDirecList\* nodetail = tail->pre;

        biDirecList\* nodeNewTail = nodetail->pre;

        nodeNewTail->next = tail;

        tail->pre = nodeNewTail;

    }

    void removeNode(biDirecList\* node){

        biDirecList\* preNode = node->pre;

        biDirecList\* nextNode = node->next;

        preNode->next = nextNode;

        nextNode->pre = preNode;

    }

    void moveToHead(biDirecList\* node){

        removeNode(node);

        addHead(node);

    }

};

* 1. 前K个高频元素

给你一个整数数组 nums 和一个整数 k ，请你返回其中出现频率前 k 高的元素。你可以按 **任意顺序** 返回答案。

【解题思路】先用一个map存储所有数字的出现次数，然后用一个小顶堆筛选出前K个频率的元素。

class Solution {

public:

    struct cmp{

        bool operator()(pair<int,int> a,pair<int,int> b){

            return a.second > b.second;//小顶堆是大于号

        }

    };

    vector<int> topKFrequent(vector<int>& nums, int k) {

        unordered\_map<int,int> cmap;

        for(auto &n:nums){

            cmap[n]++;

        }

        priority\_queue<pair<int,int>,vector<pair<int,int>>,cmp> que;

        for(auto &pair:cmap){

            que.push(pair);

            if(que.size()>k){

                que.pop();

            }

        }

        vector<int> ans;

        while(!que.empty()){

            ans.emplace\_back(que.top().first);

            que.pop();

        }

        return ans;

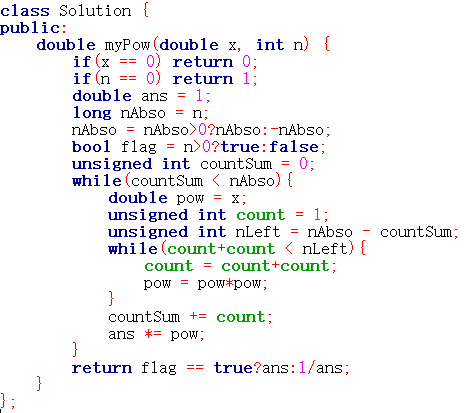
    }

};

1. 快速幂
   1. 数值的整数次方

实现 [pow(x, n)](https://www.cplusplus.com/reference/valarray/pow/) ，即计算 x 的 n 次幂函数（即，xn）。不得使用库函数，同时不需要考虑大数问题。

【解题思路】快速幂法时间复杂度为log(n)，注意int 的溢出问题。



1. 有限状态自动机
   1. 表示数值的字符串

请实现一个函数用来判断字符串是否表示数值（包括整数和小数）。

数值（按顺序）可以分成以下几个部分：

1. 若干空格
2. 一个 小数 或者 整数
3. （可选）一个 'e' 或 'E' ，后面跟着一个 整数
4. 若干空格

小数（按顺序）可以分成以下几个部分：

1. （可选）一个符号字符（'+' 或 '-'）
2. 下述格式之一：

至少一位数字，后面跟着一个点 '.'

至少一位数字，后面跟着一个点 '.' ，后面再跟着至少一位数字

一个点 '.' ，后面跟着至少一位数字

整数（按顺序）可以分成以下几个部分：

（可选）一个符号字符（'+' 或 '-'）

至少一位数字

部分数值列举如下：

["+100", "5e2", "-123", "3.1416", "-1E-16", "0123"]

部分非数值列举如下：

["12e", "1a3.14", "1.2.3", "+-5", "12e+5.4"]

**预备知识**

确定有限状态自动机（以下简称「自动机」）是一类计算模型。它包含一系列状态，这些状态中：

有一个特殊的状态，被称作「初始状态」。

还有一系列状态被称为「接受状态」，它们组成了一个特殊的集合。其中，一个状态可能既是「初始状态」，也是「接受状态」。

起初，这个自动机处于「初始状态」。随后，它顺序地读取字符串中的每一个字符，并根据当前状态和读入的字符，按照某个事先约定好的「转移规则」，从当前状态转移到下一个状态；当状态转移完成后，它就读取下一个字符。当字符串全部读取完毕后，如果自动机处于某个「接受状态」，则判定该字符串「被接受」；否则，判定该字符串「被拒绝」。

**注意：**如果输入的过程中某一步转移失败了，即不存在对应的「转移规则」，此时计算将提前中止。在这种情况下我们也判定该字符串「被拒绝」。

一个自动机，总能够回答某种形式的「对于给定的输入字符串 S，判断其是否满足条件 P」的问题。在本题中，条件 P 即为「构成合法的表示数值的字符串」。

自动机驱动的编程，可以被看做一种暴力枚举方法的延伸：它穷尽了在任何一种情况下，对应任何的输入，需要做的事情。

自动机在计算机科学领域有着广泛的应用。在算法领域，它与大名鼎鼎的字符串查找算法「KMP」算法有着密切的关联；在工程领域，它是实现「正则表达式」的基础。

**问题描述**

在 C++ 文档 中，描述了一个合法的数值字符串应当具有的格式。具体而言，它包含以下部分：

1. 符号位，即 ++、-− 两种符号
2. 整数部分，即由若干字符 0-90−9 组成的字符串
3. 小数点
4. 小数部分，其构成与整数部分相同
5. 指数部分，其中包含开头的字符 \text{e}e（大写小写均可）、可选的符号位，和整数部分

相比于 C++ 文档而言，本题还有一点额外的不同，即允许字符串首末两端有一些额外的空格。

在上面描述的五个部分中，每个部分都不是必需的，但也受一些额外规则的制约，如：

如果符号位存在，其后面必须跟着数字或小数点。

小数点的前后两侧，至少有一侧是数字。

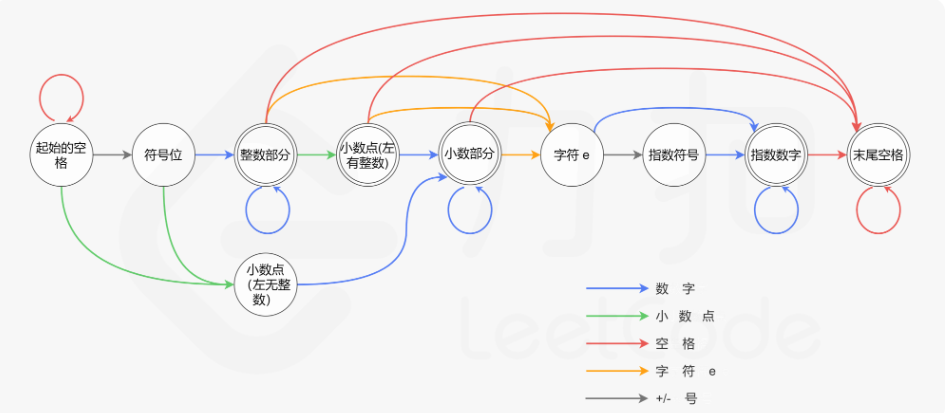
**思路与算法**

根据上面的描述，现在可以定义自动机的「状态集合」了。那么怎么挖掘出所有可能的状态呢？一个常用的技巧是，用「当前处理到字符串的哪个部分」当作状态的表述。根据这一技巧，不难挖掘出所有状态：

1. 起始的空格
2. 符号位
3. 整数部分
4. 左侧有整数的小数点
5. 左侧无整数的小数点（根据前面的第二条额外规则，需要对左侧有无整数的两种小数点做区分）
6. 小数部分
7. 字符 \text{e}e
8. 指数部分的符号位
9. 指数部分的整数部分
10. 末尾的空格

下一步是找出「初始状态」和「接受状态」的集合。根据题意，「初始状态」应当为状态 1，而「接受状态」的集合则为状态 3、状态 4、状态 6、状态 9 以及状态 10。换言之，字符串的末尾要么是空格，要么是数字，要么是小数点，但前提是小数点的前面有数字。

最后，需要定义「转移规则」。结合数值字符串应当具备的格式，将自动机转移的过程以图解的方式表示出来：



class Solution {

public:

    enum State {

        STATE\_INITIAL,

        STATE\_INT\_SIGN,

        STATE\_INTEGER,

        STATE\_POINT,

        STATE\_POINT\_WITHOUT\_INT,

        STATE\_FRACTION,

        STATE\_EXP,

        STATE\_EXP\_SIGN,

        STATE\_EXP\_NUMBER,

        STATE\_END

    };

    enum CharType {

        CHAR\_TYPE\_INTEGER,

        CHAR\_TYPE\_DOT,

        CHAR\_TYPE\_SIGN,

        CHAR\_TYPE\_E,

        CHAR\_TYPE\_SPACE,

        CHAR\_TYPE\_ILLEGAL

    };

    CharType toCharType(char c){

        if((c>='0')&&(c<='9'))

            return CHAR\_TYPE\_INTEGER;

        else if(c == '.')

            return CHAR\_TYPE\_DOT;

        else if(c == 'e'||c == 'E')

            return CHAR\_TYPE\_E;

        else if(c == '+'||c == '-')

            return CHAR\_TYPE\_SIGN;

        else if(c == ' ')

            return CHAR\_TYPE\_SPACE;

        else

            return CHAR\_TYPE\_ILLEGAL;

    }

    bool isNumber(string s) {

        unordered\_map<State,unordered\_map<CharType,State>> transfer =

        {

            {

                STATE\_INITIAL, {

                    {CHAR\_TYPE\_INTEGER,STATE\_INTEGER},

                    {CHAR\_TYPE\_SPACE,STATE\_INITIAL},

                    {CHAR\_TYPE\_SIGN,STATE\_INT\_SIGN},

                    {CHAR\_TYPE\_DOT,STATE\_POINT\_WITHOUT\_INT}

                }

            },

            {

                STATE\_INTEGER, {

                    {CHAR\_TYPE\_INTEGER,STATE\_INTEGER},

                    {CHAR\_TYPE\_DOT,STATE\_FRACTION},

                    {CHAR\_TYPE\_E,STATE\_EXP},

                    {CHAR\_TYPE\_SPACE,STATE\_END}

                }

            },

            {

                STATE\_POINT,{

                    {CHAR\_TYPE\_INTEGER,STATE\_FRACTION},

                    {CHAR\_TYPE\_E,STATE\_EXP},

                    {CHAR\_TYPE\_SPACE,STATE\_END}

                }

            },

            {

                STATE\_INT\_SIGN,{

                    {CHAR\_TYPE\_INTEGER,STATE\_INTEGER},

                    {CHAR\_TYPE\_DOT,STATE\_POINT\_WITHOUT\_INT}

                }

            },

            {

                STATE\_POINT\_WITHOUT\_INT,{

                    {CHAR\_TYPE\_INTEGER,STATE\_FRACTION}

                }

            },

            {

                STATE\_FRACTION,{

                    {CHAR\_TYPE\_INTEGER,STATE\_FRACTION},

                    {CHAR\_TYPE\_E,STATE\_EXP},

                    {CHAR\_TYPE\_SPACE,STATE\_END}

                }

            },

            {

                STATE\_EXP,{

                    {CHAR\_TYPE\_SIGN,STATE\_EXP\_SIGN},

                    {CHAR\_TYPE\_INTEGER,STATE\_EXP\_NUMBER}

                }

            },

            {

                STATE\_EXP\_SIGN,{

                    {CHAR\_TYPE\_INTEGER,STATE\_EXP\_NUMBER}

                }

            },

            {

                STATE\_EXP\_NUMBER,{

                    {CHAR\_TYPE\_INTEGER,STATE\_EXP\_NUMBER},

                    {CHAR\_TYPE\_SPACE,STATE\_END}

                }

            },

            {

                STATE\_END,{

                    {CHAR\_TYPE\_SPACE,STATE\_END}

                }

            }

        };

        State state = STATE\_INITIAL;

        for(auto c:s)

        {

            CharType type = toCharType(c);

            //cout<<type;

            auto map = transfer[state];

            if(map.find(type) == map.end())

                return false;

            state = map[type];

        }

        return (state == STATE\_INTEGER||state == STATE\_FRACTION||state == STATE\_POINT||state == STATE\_EXP\_NUMBER||state == STATE\_END);

    }

};

1. 排序
   1. 堆排序

* 首先将数组构造成堆

从最后一个非叶子节点开始（array.size()/2 -1）从左至右，从下至上进行上浮操作。

* 不断的执行取出堆顶元素的操作。

每次将堆顶元素和最后一个元素交换位置，然后执行下沉操作。

* 1. 快排

快排的思想是每次选组一个数，将小于它的数放置在它的左边，将大于它的数放置在它的右边，递归进行此项操作。

* 刚开始，i和j分别指向数组头和数组尾，基准取第一个数，然后从位置j开始向左寻找比index小的数，找到之后将该数字填到array[i]，然后将i向右移动一个位置。
* 然后从位置i开始向右寻找比index大的数，找到后将该值填到刚刚空出来的坑里，并将j向左移动一个位置。
* 重复上述过程，直到i = j

void quickSort(int left, int right, vector<int>& arr){

        if(left >= right){

            return;

        }

**int pivot = rand() % (right - left + 1) + left;//时间复杂度优化，随机取一个数字与第一个数字交换**

**swap(arr[left],arr[pivot]);**

        int i = left, j = right;

        int base = arr[i];

        while(i<j){

            while(arr[j]>=base&&i<j) j--;

            arr[i] = arr[j];

            while(arr[i]<=base&&i<j) i++;

            arr[j] = arr[i];

        }

        arr[i] = base;

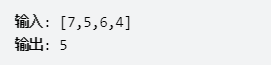
        quickSort(left,i-1,arr);

        quickSort(i+1,right,arr);

}

* 1. 组中的逆序对

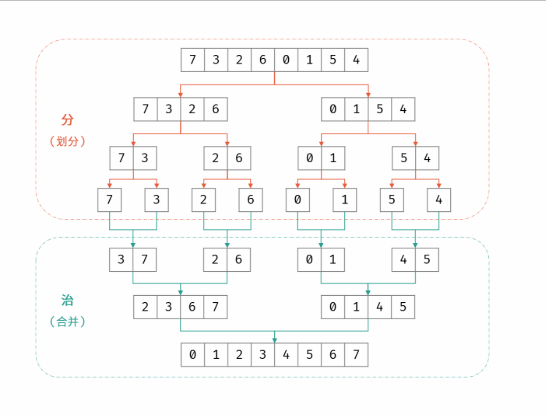
在数组中的两个数字，如果前面一个数字大于后面的数字，则这两个数字组成一个逆序对。输入一个数组，求出这个数组中的逆序对的总数。



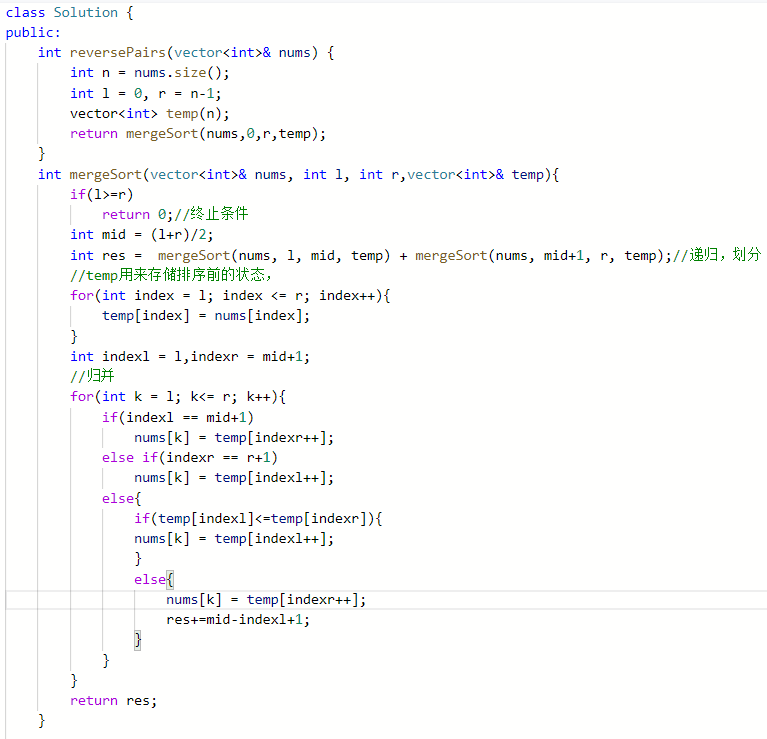
【解题思路】

分： 不断将数组从中点位置划分开（即二分法），将整个数组的排序问题转化为子数组的排序问题；

治： 划分到子数组长度为 1 时，开始向上合并，不断将 较短排序数组 合并为 较长排序数组，直至合并至原数组时完成排序。



合并阶段 本质上是 合并两个排序数组 的过程，而每当遇到 左子数组当前元素 > 右子数组当前元素 时，意味着 「左子数组当前元素 至 末尾元素」 与 「右子数组当前元素」 构成了若干 「逆序对」 。



* 1. 数组中的第K个最大元素

给定整数数组 nums 和整数 k，请返回数组中第 **k** 个最大的元素。

请注意，你需要找的是数组排序后的第 k 个最大的元素，而不是第 k 个不同的元素。

【解题思路】利用快排找到第K个数字就返回。

class Solution {

public:

    int findKthLargest(vector<int>& nums, int k) {

        if(nums.size() == 0) return 0;

        return quickSort(nums, 0, nums.size()-1,k);

    }

    int quickSort(vector<int>& nums, int left, int right,int k){

        int index = nums[left];

        int i = left,j = right;

        while(i<j){

            while(i<j && nums[j] <= index){

                j--;

            }

            if(i < j){

                nums[i++] = nums[j];

            }

            while(i<j && nums[i] > index){

                i++;

            }

            if(i < j){

                nums[j--] = nums[i];

            }

        }

        nums[i] = index;

        if(i == k-1)

            return nums[i];

        else if(i > k-1)

            return quickSort(nums,left,i-1,k);

        else

            return quickSort(nums,i+1,right,k);

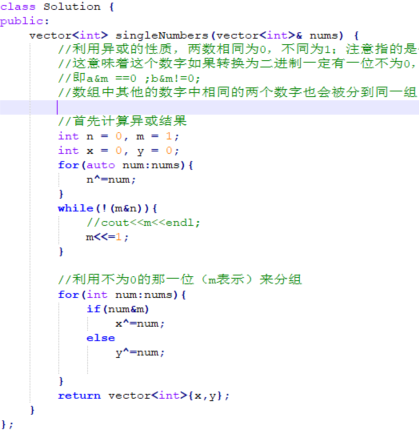
    }

};

1. 位运算
   1. 找出这数组中两个只出现一次的数字

一个整型数组 nums 里除两个数字之外，其他数字都出现了两次。请写程序找出这两个只出现一次的数字。要求时间复杂度是O(n)，空间复杂度是O(1)。

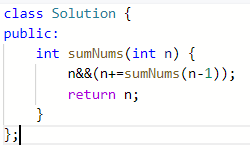
【解题思路】利用异或的性质，两数相同为0，不同为1；注意指的是位运算，两个十进制数字异或会得到一个不为0的数，这意味着这个数字如果转换为二进制一定有一位不为0，那么就可以利用这一位将两个数字分到两个不同的组，即a&m ==0 ;b&m!=0;数组中其他的数字中相同的两个数字也会被分到同一组。



* 1. 求1+2+…+n

求 1+2+...+n ，要求不能使用乘除法、for、while、if、else、switch、case等关键字及条件判断语句（A?B:C）。

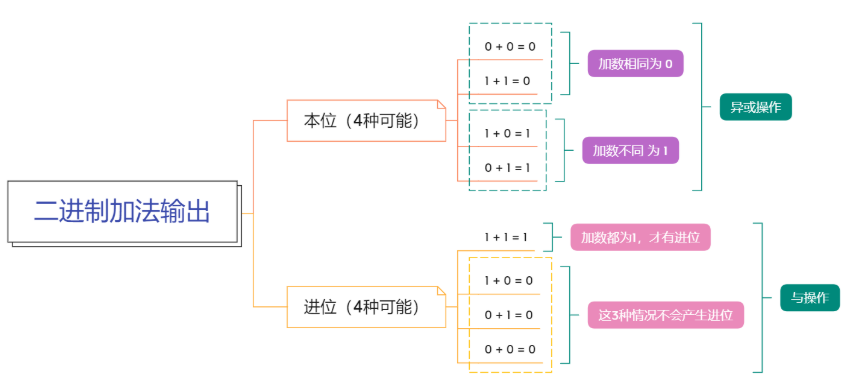
【解题思路】递归，递归终点用&&来计算。

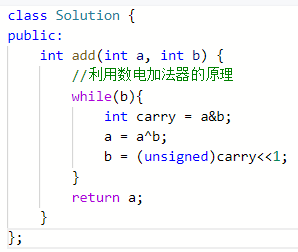


* 1. 不用加减乘除做加法

写一个函数，求两个整数之和，要求在函数体内不得使用 “+”、“-”、“\*”、“/” 四则运算符号。

【解题思路】利用数电加法器的原理，利用异或来计算本位，利用&来计算进位，然后进位左移再与上一轮计算出来的本位计算新一轮的进位和本位，直到计算出来的进位为0





1. 单调队列
   1. 滑动窗口的最大值

给定一个数组 nums 和滑动窗口的大小 k，请找出所有滑动窗口里的最大值。



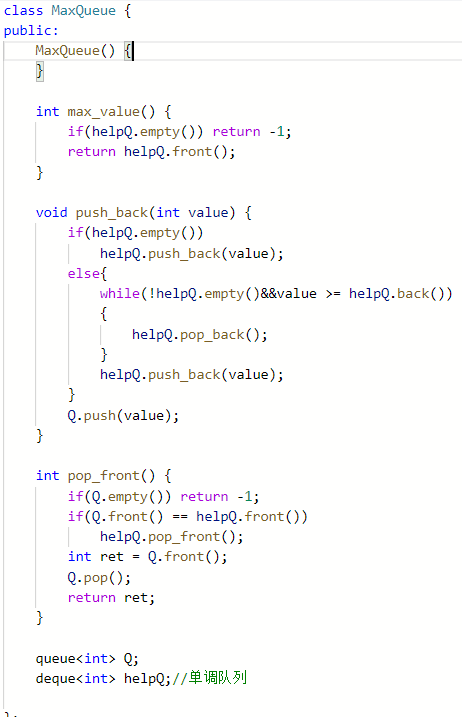
【解题思路】**单调队列，**里面的数据由大到小排列，随着窗口的滑动进行更新，里面的一个数据永远是窗口内的最大值。随着窗口的移动，不断地将新的元素与队尾的元素比较，如果前者大于后者，队尾的元素将被永远的移除，重复此项操作直到队列为空或新的元素小于队尾的元素。

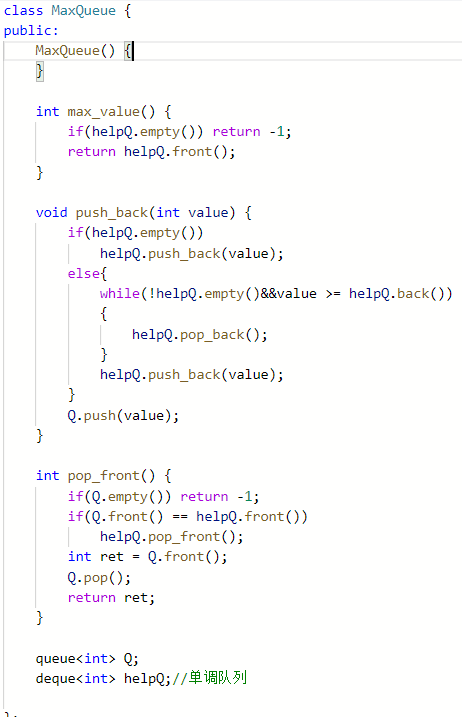


* 1. 队列的最大值

请定义一个队列并实现函数 max\_value 得到队列里的最大值，要求函数max\_value、push\_back 和 pop\_front 的均摊时间复杂度都是O(1)。

若队列为空，pop\_front 和 max\_value 需要返回 -1

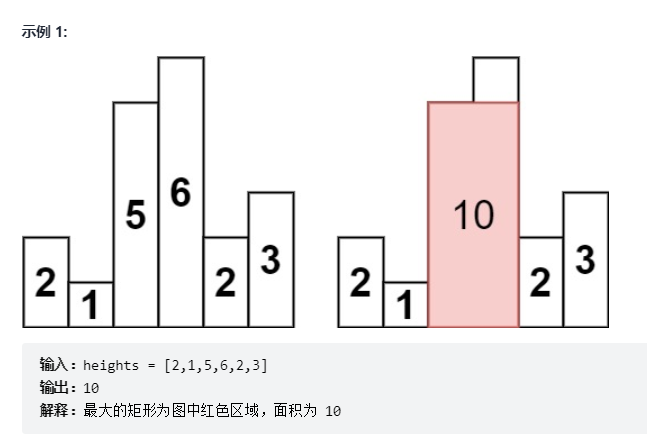




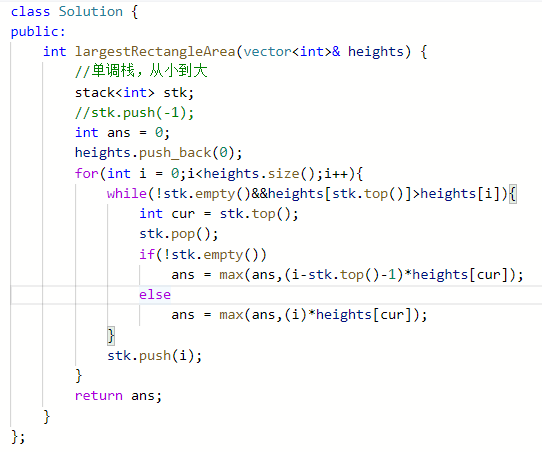
* 1. 柱状图中最大矩形

给定 n 个非负整数，用来表示柱状图中各个柱子的高度。每个柱子彼此相邻，且宽度为 1 。

求在该柱状图中，能够勾勒出来的矩形的最大面积。



【解题思路】对于每一个柱子，分别向左和向右找到第一个比它低的柱子，计算出以它为高度的矩形面积。以柱子为例，向左比它低的柱子是1，右边比它低的柱子是2，因此以5为高度的矩形面积为（4-1-1）\*5 = 10。考虑用单调栈来记录左边最近的比他低的柱子，栈中数据递增排列，遇到比top大的数字直接push，遇到比top小的数据需要把所有比该数据小的数据都pop出来，再push。因此对于每一个柱子而言，左边第一个比它小的数据是把它自己pop出去后栈顶的数据，右边第一个比它小的数据是还没有进到栈中的heights[i]



* 1. 接雨水

给定 n 个非负整数表示每个宽度为 1 的柱子的高度图，计算按此排列的柱子，下雨之后能接多少雨水。

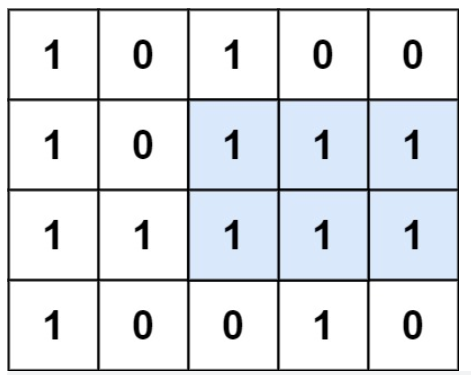


【解题思路】思路与柱状图中的最大矩形相似，对于每个柱子，获取左边和右边第一个比它大的柱子位置，单调栈从大到小排列，被pop出去时获得右边的最大值，pop出去之后栈中的top是左边最大值。



* 1. 最大矩形

给定一个仅包含 0 和 1 、大小为 rows x cols 的二维二进制矩阵，找出只包含 1 的最大矩形，并返回其面积。



【解题思路】将矩阵的每一层都抽象成一个坐标轴，竖向连续的1的个数为柱形图的高度，可以通过求解每一层的最大柱形面积来得到答案。



* 1. 每日温度

给定一个整数数组 temperatures ，表示每天的温度，返回一个数组 answer ，其中 answer[i] 是指在第 i 天之后，才会有更高的温度。如果气温在这之后都不会升高，请在该位置用 0 来代替。

【解题思路】单调栈，从大到小排列（如果是获取右边比它大的数，就从大到小排列，如果是获取右边比它小的数，就从小到大排列），被pop出去的时候获取到右边第一个比它大的值。

class Solution {

public:

    vector<int> dailyTemperatures(vector<int>& temperatures) {

        stack<int> helps;

        vector<int> ans(temperatures.size(),0);

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

            if(helps.empty()||temperatures[i]<=temperatures[helps.top()]){

                helps.push(i);

            }

            if(temperatures[i]>temperatures[helps.top()]){

                while(!helps.empty()&&temperatures[i]>temperatures[helps.top()]){

                    int temp = helps.top();

                    ans[temp] = i-temp;

                    helps.pop();

                }

                helps.push(i);

            }

        }

        return ans;

    }

};

1. 前缀树

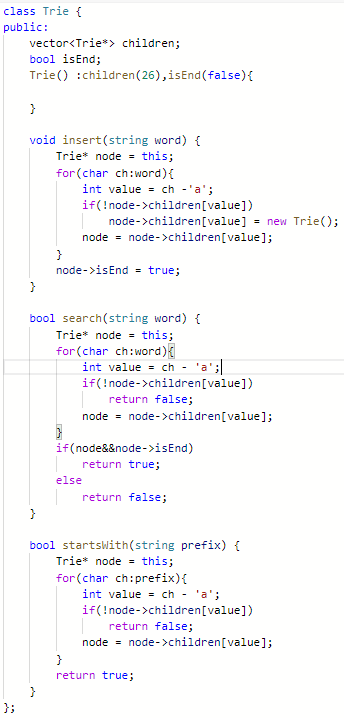
Trie（发音类似 "try"）或者说 前缀树 是一种树形数据结构，用于高效地存储和检索字符串数据集中的键。这一数据结构有相当多的应用情景，例如自动补完和拼写检查。

请你实现 Trie 类：

* Trie() 初始化前缀树对象。
* void insert(String word) 向前缀树中插入字符串 word 。
* boolean search(String word) 如果字符串 word 在前缀树中，返回 true（即，在检索之前已经插入）；否则，返回 false 。
* boolean startsWith(String prefix) 如果之前已经插入的字符串 word 的前缀之一为 prefix ，返回 true ；否则，返回 false 。

【解题方法】实质上是一种多叉树，包含一个子节点数组，用每个节点的index表示信息。例如用来表示字符串时，children[ch-‘a’]表示字符ch是否存在，数组中有些节点为空，表示没有这个index所表示的信息。前缀树需要实现几个函数：

* void insert(string word)
* bool search(string word)
* bool startwith(string prefix)



1. Union-Find并查集

基本数据结构是一个数组，用数组的index表示每个节点，数组值表示该节点(index)的parent节点，对于根节点的特性parent[node] = node来判断该节点是否为根节点，层层迭代寻找，就可以找到每一个节点的根节点，如果两个节点根节点相同就可以判定这两个节点相连通。需要实现几个函数，

* void union2Node(int p, int q)//用来连通两个节点，方法是把节点p的根节点挂在节点q的根节点下面。或者反过来。
* int find(in p)用来寻找一个节点的根节点
* bool isConnect(int p, int q)判断两个节点是否连通

class UnionFind {

    vector<int>parent;

public:

    UnionFind(int size){

        parent = vector<int>(size);

        for(int i = 0;i<size;i++){

            parent[i] = i;

        }

    }

    void union2Node(int p, int q){

        int rootP = find(p);

        int rootQ = find(q);

        if(rootP == rootQ)

            return;

        parent[rootP] = rootQ;

    }

    int find(int p ){

        while(parent[p] != p){

            parent[p] = parent[parent[p]];//用来降低树的高度，减少迭代次数

            p = parent[p];

        }

        return p;

    }

    bool isConnect(int p, int q){

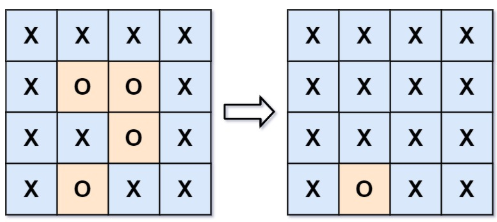
        return find(p) == find(q);

    }

};

* 1. 被围绕的区域

给你一个 m x n 的矩阵 board ，由若干字符 'X' 和 'O' ，找到所有被 'X' 围绕的区域，并将这些区域里所有的 'O' 用 'X' 填充。被围绕的区间不会存在于边界上，换句话说，任何边界上的 'O' 都不会被填充为 'X'。 任何不在边界上，或不与边界上的 'O' 相连的 'O' 最终都会被填充为 'X'。如果两个元素在水平或垂直方向相邻，则称它们是“相连”的。



【解题思路】首先把所有外围的O都连接到一个outter根节点上，然后遍历内部的节点，构建并查集。然后再次遍历内部节点，把根节点不是outter的节点改为‘X’

class Solution {

public:

    vector<vector<int>> direc;

    void solve(vector<vector<char>>& board) {

        if(board.size() == 0) return;

        direc = {{1,0},{-1,0},{0,1},{0,-1}};

        int m = board.size();

        int n = board[0].size();

        UnionFind unin(m\*n+1);

        //UnionFind \*p = new UnionFind(m\*n+1);

        int outter = m\*n;

        for(int i = 0;i<m;i++){

            if(board[i][0] == 'O')

            {

                unin.union2Node(i\*n,outter);

            }

            if(board[i][n-1] == 'O')

            {

                unin.union2Node(i\*n+n-1,outter);

            }

        }

        for(int j = 0;j<n;j++){

            if(board[0][j] == 'O')

                unin.union2Node(j,outter);

            if(board[m-1][j] == 'O')

                unin.union2Node((m-1)\*n+j,outter);

        }

        for(int i = 1;i<m-1;i++){

            for(int j = 1;j<n-1;j++){

                if(board[i][j] == 'O'){

                    for(auto di:direc)

                        if(board[i+di[0]][j+di[1]] == 'O'){

                            //cout<<i\*n+j<<endl;

                            unin.union2Node((i+di[0])\*n+j+di[1],i\*n+j);

                        }

                }

            }

        }

        for(int i = 0;i<m-1;i++){

            for(int j = 0;j<n-1;j++){

                if(!unin.isConnect(outter,i\*n+j)){

                    board[i][j] = 'X';

                }

            }

        }

    }

};

1. 前缀和
   1. 和为K的子数组

给你一个整数数组 nums 和一个整数 k ，请你统计并返回该数组中和为 k的连续子数组的个数。

【解题思路】因为数组里面既有正数也有负数，所以无法利用滑动窗口获得窗口收缩条件，因此采用前缀和+哈希表算法。利用哈希表存储所有的前缀和（key）与出现次数（value），并通过更新变量pre记录每个下标位置的前缀和，对于每个下标位置的元素，它的前缀和为pre，需要获取在它前面并且前缀和为pre-k的下标个数，即为hashmap[pre-k]。

class Solution {

public:

    int subarraySum(vector<int>& nums, int k) {

        unordered\_map<int,int> hashmap;

        hashmap[0] = 1;

        int pre = 0;

        int count = 0;

        for(auto& n:nums){

            pre+=n;

            if(hashmap.find(pre-k)!=hashmap.end()){

                count += hashmap[pre-k];

            }

            hashmap[pre]++;

        }

        return count;

    }

};

1. 不知道如何分类（字符串，递归等）
   1. 字符串解码

给定一个经过编码的字符串，返回它解码后的字符串。

编码规则为: k[encoded\_string]，表示其中方括号内部的 encoded\_string 正好重复 k 次。注意 k 保证为正整数。

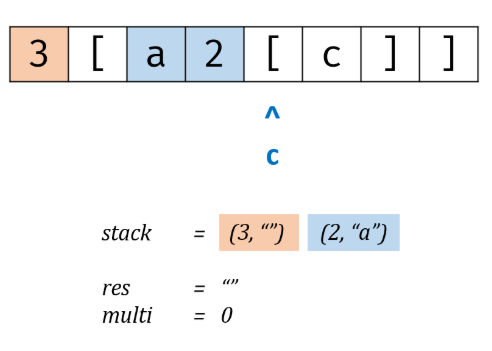
你可以认为输入字符串总是有效的；输入字符串中没有额外的空格，且输入的方括号总是符合格式要求的。

此外，你可以认为原始数据不包含数字，所有的数字只表示重复的次数 k ，例如不会出现像 3a 或 2[4] 的输入。

**【解题思路】**

方法1：递归，遇到”[“进入递归，遇到”]”或到达字符串末端递归结束。

方法2：辅助栈，当ch为数字时，计算数值，当ch为字母时，append到res；当ch为”[“时，将multi和res入栈，并将二者清零，当ch为“]“为出栈获取last\_res和multi，更新res = last\_res+multi\*res。



class Solution {

public:

    stack<pair<int,int>> helps;//first:repeattime;second:[index

    string ans;

    int l;

    string decodeString(string s) {

        l = 0;

        while(l<s.size()){

            ans += helpDecode(s);

            //l++;

        }

        return ans;

    }

    string helpDecode(string s){

        int lnew = 0,rnew = 0,intIndex = 0;

        string retStr;

        int repeatTime = 0;

        while(l<s.size()){

            while(l<s.size()&&s[l]>='0'&&s[l]<='9'){

                repeatTime = repeatTime\*10+s[l]-'0';

                l++;

            }

            if(s[l] == '['){

                l++;

                string inStr = helpDecode(s);

                while(repeatTime>0) {

                    retStr+=inStr;

                    repeatTime--;

                }

                continue;

            }

            if(s[l] == ']')

            {

                l++;

                return retStr;

            }

            retStr += s[l];

            l++;

        }

        return retStr;

    }

};

* 1. 合并区间

以数组 intervals 表示若干个区间的集合，其中单个区间为 intervals[i] = [starti, endi] 。请你合并所有重叠的区间，并返回 一个不重叠的区间数组，该数组需恰好覆盖输入中的所有区间 。

【解题思路】我们将列表中的区间按照左端点升序排序。然后我们将第一个区间加入 merged 数组中，并按顺序依次考虑之后的每个区间：

如果当前区间的左端点在数组 merged 中最后一个区间的右端点之后，那么它们不会重合，我们可以直接将这个区间加入数组 merged 的末尾；

否则，它们重合，我们需要用当前区间的右端点更新数组 merged 中最后一个区间的右端点，将其置为二者的较大值

class Solution {

public:

    vector<vector<int>> merge(vector<vector<int>>& intervals) {

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

        vector<vector<int>>ans;

        ans.push\_back(intervals[0]);

        for(auto pair:intervals){

            if(pair[0]<=ans[ans.size()-1][1])

                ans[ans.size()-1][1] = max(pair[1],ans[ans.size()-1][1]);

            else

                ans.push\_back(pair);

        }

        return ans;

    }

};