常用算法

# 贪心

## [盛最多水的容器](https://leetcode-cn.com/problems/container-with-most-water)

矮边是乘积的成员，而宽度是一直减少的，贪心移动矮边

class Solution {

public:

    int maxArea(vector<int>& height) {

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

        int nMax = 0;

        while(left<right){

            nMax = max(nMax, min(height[left], height[right])\*(right-left));

            if(height[left]<height[right])left++;

            else right--;

        }

        return nMax;

    }

};

[摆动序列](https://leetcode-cn.com/problems/wiggle-subsequence)

方向不同的才计数

class Solution {

public:

    int wiggleMaxLength(vector<int>& nums)

    {

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

        int op = 0, g = 1;

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

            if (nums[i] > nums[i - 1])

            {

                if (op <= 0)g++;

                op = 1;

            }

            else if (nums[i] < nums[i - 1])

            {

                if (op >= 0)g++;

                op = -1;

            }

        }

        return g;

    }

};

## [最接近的三数之和](https://leetcode-cn.com/problems/3sum-closest)

class Solution {

public:

    int threeSumClosest(vector<int>& nums, int target)

    {

        if (nums.size() < 3)return 0;

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

        int res(0),diff(INT\_MAX);

        for(int i = 0;i < nums.size()-2;++i)

        {

            int left = i + 1,right = nums.size()-1;

            while(left < right)//贪心遍历

            {

                int s = nums[i] + nums[left] + nums[right];

                if (s == target)return s;

                int d = abs(target - s);

                if (d < diff)

                {

                    res = s;

                    diff = d;

                }

                if (s < target)++left;

                else --right;

            }

        }

        return res;

    }

};

## [搜索二维矩阵 II](https://leetcode-cn.com/problems/search-a-2d-matrix-ii)

class Solution {

public:

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

        if (matrix.size() == 0 || matrix[0].size() == 0)return false;

        int m = matrix.size();

        int n = matrix[0].size();

        int x = m - 1;

        int y = 0;

        for(;x >= 0 && y < n;)

        {

            if (matrix[x][y] < target)//贪心

            {

                ++y;

            }

            else if (matrix[x][y] > target)

            {

                --x;

            }

            else return true;

        }

        return false;

    }

};

## [最长连续序列](https://leetcode-cn.com/problems/longest-consecutive-sequence)

class Solution {

public:

    int longestConsecutive(vector<int>& nums)

    {

        if (nums.size() <1)return nums.size();

        unordered\_set<int> st(nums.begin(),nums.end());//哈希表

        int res(0),left,right;

        for(auto n:nums)

        {

            if (st.count(n - 1) == 0)//只计算左边界开始的

            {

                right = left = n;

                while(st.count(right+1))//贪心算法

                {

                    ++right;

                }

                res = max(res,right - left + 1);

            }

        }

        return res;

    }

};

## [最大子序和](https://leetcode-cn.com/problems/maximum-subarray)

class Solution {

public:

    int maxSubArray(vector<int>& nums)

    {

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

        int res = nums[0],sum(0);

        for(auto& n:nums)

        {

            sum += n;

            res = max(res,sum);//贪心

            if (sum < 0)sum = 0;//断开不需要的

        }

        return res;

    }

};

# 分治

## [单词拆分 II](https://leetcode-cn.com/problems/word-break-ii)

class Solution {

public:

    unordered\_map<string,vector<string>> m;//复杂度 K \* n

    vector<string> wordBreak(string s, vector<string>& wordDict)

    {

        if (s.size() == 0)return {""};//空的为收敛条件

        if (m.count(s))return m[s];

        vector<string> res;

        for(auto& w:wordDict)

        {

            if (s.substr(0,w.size()) == w)

            {

                vector<string> subv = wordBreak(s.substr(w.size()),wordDict);//分治

                for(auto& t:subv)

                {

                     res.push\_back(t.size() ? w + string(" ") + t : w + t);

                }

            }

        }

        return m[s] = res;//从后往前收敛，数组已是本子串的所有情况

    }

};

## [扁平化嵌套列表迭代器](https://leetcode-cn.com/problems/flatten-nested-list-iterator)

## [为运算表达式设计优先级](https://leetcode-cn.com/problems/different-ways-to-add-parentheses)

# dfs

（排列）结束的条件为遍历的长度

## [全排列](https://leetcode-cn.com/problems/permutations)

选择一个，递归，恢复原来，不需要检查去重

class Solution {

public:

    vector<vector<int>> res;

    vector<vector<int>> permute(vector<int>& nums)

    {

        dfs(nums,0);

        return res;

    }

    //不含重复数组

    void dfs(vector<int>& nums,int pos)

    {

        if (pos == nums.size())

        {

            res.push\_back(nums);

            return;

        }

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

        {

            swap(nums[pos],nums[i]);

            dfs(nums,pos+1);

            swap(nums[pos],nums[i]);

        }

    }

};

## [全排列 II](https://leetcode-cn.com/problems/permutations-ii)

选择一个，利用有序性去重，继续选下一个，为了保持顺序而不恢复，但递归需要拷贝

class Solution {

    vector<vector<int>> res;

public:

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

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

        dfs(nums, 0);

        return res;

    }

    //含重复数组，结果序列需要不重复

    void dfs(vector<int> nums, int pos) {//拷贝数组，为了不影响上层的数组

        if (pos == nums.size())

            res.push\_back(nums);

        else {

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

                if (i != pos && nums[pos] == nums[i]) continue;   //不与相同的交换

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

                dfs(nums, pos + 1);

            }

        }

    }

};

## [组合](https://leetcode-cn.com/problems/combinations)

结束的条件为达到目标，比如数量

路径为

class Solution {

public:

    vector<vector<int>> vv;

    vector<int> v;

    vector<vector<int>> combine(int n, int k) {

        vv.clear();

        v.clear();

        dfs(1,n,k);

        return vv;

    }

    void dfs(int pos,int n, int k)

    {

        for(int i = pos;i <= n;++i)

        {

            v.push\_back(i);//选择

            if (v.size() == k)vv.push\_back(v);

            else dfs(i + 1,n,k);

            v.pop\_back();//不选择，需要放后面

        }

    }

};

## [子集](https://leetcode-cn.com/problems/subsets)

dfs，选择或者不选择，没去重

class Solution {

public:

    vector<vector<int>> res;

    vector<int> path;

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

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

        res.push\_back(path);

        dfs(0,nums);

        return res;

    }

    void dfs(int pos ,vector<int>& nums)

    {

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

        {

            path.push\_back(nums[i]);

            res.push\_back(path);

            dfs(i+1,nums);

            path.pop\_back();

        }

    }

};

## [子集 II](https://leetcode-cn.com/problems/subsets-ii)

dfs，利用有序性去重，去重在后面执行，因为需要先遍历

class Solution {

public:

    vector<int> path;

    vector<vector<int>> result;

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

        path.clear();

        result.clear();

        result.push\_back(path);

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

        dfs(nums,0,path,result);

        return result;

    }

    void dfs(vector<int> &nums,int pos,vector<int>& path,vector<vector<int>> &result)

    {

        if(pos==nums.size())

            return ;

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

        {

            path.push\_back(nums[i]);

            result.push\_back(path);

            dfs(nums,i+1,path,result);

            path.pop\_back();

            while(nums[i]==nums[i+1]) i++;//开始位置不用重复的数

        }

    }

};

## [组合总和](https://leetcode-cn.com/problems/combination-sum)

需要从小到大处理，因为使用了条件过滤（target >= candidates[i]）

class Solution {

public:

    vector<vector<int>> results;

    vector<int> result;

    vector<vector<int>> combinationSum(vector<int>& candidates, int target) {

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

        dfs(candidates,target,0);

        return results;

    }

    void dfs(vector<int>& candidates,int target, int index){

        if(target == 0){

            results.push\_back(result);

            return;

        }

        for(int i = index; i < candidates.size() && target >= candidates[i]; ++i)

        {

            result.push\_back(candidates[i]);

            dfs(candidates,target - candidates[i], i);

            result.pop\_back();

        }

    }

};

或者

class Solution {

public:

    vector<vector<int>> results;

    vector<int> result;

    vector<vector<int>> combinationSum(vector<int>& candidates, int target) {

        //sort(candidates.begin(), candidates.end());

        dfs(candidates,target,0);

        return results;

    }

    void dfs(vector<int>& candidates,int target, int index){

        if(target == 0){

            results.push\_back(result);

            return;

        }

        if (target > 0)

        {

            for(int i = index; i < candidates.size(); ++i)

            {

                result.push\_back(candidates[i]);

                dfs(candidates,target - candidates[i], i);

                result.pop\_back();

            }

        }

    }

};

## [组合总和 II](https://leetcode-cn.com/problems/combination-sum-ii)

dfs,需要排序，利用有序性去重

class Solution {

public:

    vector<vector<int>> results;

    vector<int> result;

    vector<vector<int>> combinationSum2(vector<int>& candidates, int target) {

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

        dfs(candidates,target,0);

        return results;

    }

    void dfs(vector<int>& candidates,int target, int index)

    {

        if(target == 0)

        {

            results.push\_back(result);

            return;

        }

        for(int i = index; i < candidates.size() && target >= candidates[i];)//需要从小到大才能这么判断

        {

            result.push\_back(candidates[i]);//选择

            dfs(candidates,target - candidates[i], i+1);

            result.pop\_back();//不选择

            ++i;

            while (i < candidates.size() && candidates[i] == candidates[i-1])++i;//本层去重

        }

    }

};

## [组合总和 III](https://leetcode-cn.com/problems/combination-sum-iii)

（其他）

## [复原IP地址](https://leetcode-cn.com/problems/restore-ip-addresses)

## [串联所有单词的子串](https://leetcode-cn.com/problems/substring-with-concatenation-of-all-words)

## [N皇后](https://leetcode-cn.com/problems/n-queens)

dfs遍历，一行一行的去遍历，每行只能一皇后，需要检查列和斜线。

到了最后一行就是收敛条件。然后把该结果放入结果集

class Solution {

public:

    vector<vector<string>> res;

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

    {

        //每行皇后列位置，只需要记录每行的下标

        vector<int> cols(n,0);

        dfs(cols, n, 0);

        return res;

    }

private:

    void dfs(vector<int>& cols, int n, int x)

    {

        if(n == x)

        {

            vector<string> v(n, string(n, '.'));

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

            {

                v[i][cols[i]] = 'Q';

            }

            res.push\_back(v);

            return;

        }

        //一行中需要遍历所有的列

        for(cols[x] = 0; cols[x] < n; ++cols[x])

        {

            if(safe(cols, x))

            {

                dfs(cols, n, x + 1);//下一行

            }

        }

    }

    bool safe(vector<int> &cols, int x)

    {

        //检查之前的行的皇后

        for(int i = 0; i < x; ++i)

        {//列值是否相同   ,  是否在同一斜线

            if(cols[i] == cols[x] || abs(cols[x] - cols[i]) == abs(x - i))

            {

                return false;

            }

        }

        return true;

    }

};

## [括号生成](https://leetcode-cn.com/problems/generate-parentheses)

## [正则表达式匹配](https://leetcode-cn.com/problems/regular-expression-matching)

# 二分

## [有效的完全平方数](https://leetcode-cn.com/problems/valid-perfect-square)

二分尝试

class Solution {

public:

    bool isPerfectSquare(int num) {

        int left = 1,right = num;

        long mid,tmp;

        while(left <= right)

        {

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

            tmp = mid \* mid;

            if (tmp == num)return true;

            else if (tmp < num)left = mid + 1;

            else right = mid - 1;

        }

        return false;

    }

};

## [第一个错误的版本](https://leetcode-cn.com/problems/first-bad-version)

// Forward declaration of isBadVersion API.

bool isBadVersion(int version);

class Solution {

public:

    int firstBadVersion(int n) {

        if(n < 1)return -1;

        int low = 1;

        int high = n;

        int mid;

        while(low + 1< high)// +1 条件是为了保证low比high至少小1，来确认第一个bad version

        {

            mid = low + (high - low) / 2;

            if(isBadVersion(mid))

            {

                high = mid;//需要包含该bad version

            }

            else

            {

                low = mid;//需要包含该bad version

            }

        }

        if(isBadVersion(low))//需要先判断low

        {

            return low;

        }

        else if(isBadVersion(high))

        {

            return high;

        }

        return -1;

    }

};

## [搜索插入位置](https://leetcode-cn.com/problems/search-insert-position)

## [矩形区域不超过 K 的最大数值和](https://leetcode-cn.com/problems/max-sum-of-rectangle-no-larger-than-k)

## [Pow(x, n)](https://leetcode-cn.com/problems/powx-n)

# 连续序列

## [乘积最大子序列](https://leetcode-cn.com/problems/maximum-product-subarray)

# 二维数组

## [旋转图像](https://leetcode-cn.com/problems/rotate-image)

## [Z 字形变换](https://leetcode-cn.com/problems/zigzag-conversion)

# 哈希表

## [两数之和](https://leetcode-cn.com/problems/two-sum)

## [三数之和](https://leetcode-cn.com/problems/3sum)

## [四数之和](https://leetcode-cn.com/problems/4sum)

需要利用哈希表访问快，以及数组的有序性来去重

class Solution {

public:

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

        vector< vector<int> > res;

        if (nums.size() < 4) return res;

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

        unordered\_map<int, vector<pair<int, int> > > hmap;           //两数和之组合

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

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

                hmap[nums[i] + nums[j]].push\_back(pair<int, int>(i, j));

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

        {

            if (i && nums[i] == nums[i-1])continue;

            for (int j = i + 1; j < nums.size(); j ++)//b

            {

                if (j > i + 1 && nums[j] == nums[j-1])continue;

                int gap = target - nums[i] - nums[j];

                if (hmap.find(gap) != hmap.end())

                {

                    auto &vec = hmap[gap];

                    for (int k = 0; k < vec.size(); k++)//c\d

                    {

                        if (vec[k].first <= j)continue;

                        vector<int> v({nums[i], nums[j] ,nums[ vec[k].first ], nums[ vec[k].second ] });//a≤b≤c≤d

                        if (res.size() && res.back() == v)continue;

                        res.push\_back(v);

                    }

                }

            }

        }

        return res;

    }

};

## [单词模式](https://leetcode-cn.com/problems/word-pattern)

class Solution {

public:

bool wordPattern(string pattern, string str) {

unordered\_map<char, int> mP;

unordered\_map<string, int> mS;

istringstream in(str);

int i = 0;

for (string word; in >> word ; ++i)

{

if (mP.find(pattern[i]) != mP.end() || mS.find(word) != mS.end())

{

if (mP[pattern[i]] != mS[word]) //如果pat 到了末尾就是0了,也会结束

{

//printf("4)%d %s %c %d\n",i,word.c\_str(),pattern[i],mS[word]);

return false;

}

//printf("%d %s\n",i,word.c\_str());

}

else

{

mP[pattern[i]] = mS[word] = i + 1;

//printf("2)%d %s\n",i,word.c\_str());

}

}

//printf("3)%d \n",i);

return i == pattern.size();

}

};

# 动规

需要找到前面k-1个成员（或者第k-1个）与第k个成员的关系

## [单词拆分](https://leetcode-cn.com/problems/word-break)

哈希表是为了快速访问，保存动规状态才能递推

class Solution {

public:

    bool wordBreak(string s, vector<string>& wordDict)

    {

        unordered\_set<string> st(wordDict.begin(),wordDict.end());

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

        dp[0] = true;

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

        {

            for(int j = 0;j < i;++j)

            {

                if (dp[j] && st.count(s.substr(j,i-j)))

                {

                    dp[i] = true;

                    break;

                }

            }

        }

        return dp[s.size()];

    }

};

## [不同的子序列](https://leetcode-cn.com/problems/distinct-subsequences)

class Solution {

public:

/\*

跟LCS类似，用双序列动态规划解决。

1. 设计：

dp[i][j]表示从第一个字符串前i个组成的子串转换为第二个字符串前j个组成的子串共有多少种方案。

2. 递推：

s[i - 1] == t[j - 1]， 则dp[i][j] = dp[i - 1][j - 1] + dp[i - 1][j];

s[i - 1] != t[j - 1]，则dp[i][j] = dp[i - 1][j];

3. 边界条件：

dp[i][0] = 1（t没有字符只有一种方案）

4. 返回值：

dp[sz1][sz2]

\*/

int numDistinct(string s, string t) {// S 的子序列中 T 出现的个数

int sz1 = s.size(), sz2 = t.size();

vector<vector<long>> dp(sz1 + 1,vector<long>(sz2 + 1,0));

for (int i = 0; i <= sz1; ++i) {

dp[i][0] = 1;//边界条件,（t没有字符只有一种方案）

}

for (int i = 1; i <= sz1; ++i) {

for (int j = 1; j <= sz2; ++j) {

if (s[i - 1] == t[j - 1]) {

dp[i][j] = dp[i - 1][j - 1] + dp[i - 1][j];//如(aatt,aat) 2= (aat,aa) 1 + (aat,aat) 1

}

else {

dp[i][j] = dp[i - 1][j];//如,(aate,aat) 1 = (aat,aat) 1

}

}

}

return dp[sz1][sz2];

}

};

## [三角形最小路径和](https://leetcode-cn.com/problems/triangle)

## [最大正方形](https://leetcode-cn.com/problems/maximal-square)

## [爬楼梯](https://leetcode-cn.com/problems/climbing-stairs)

class Solution {

public:

int climbStairs(int n) {

if (n <= 2)return n;

int vec[3];

vec[0] = 1;vec[1] = 2;//初始化值（n 为1和2 ）

for(int i=3; i<=n; i++)//计算 n -2 次

{//地推公式为： vec[n] = vec[n-1] + vec[n-2]

vec[2] = vec[0]+vec[1];

vec[0] = vec[1];//移动变量

vec[1] = vec[2];

}

return vec[2];

}

};

## [分割回文串](https://leetcode-cn.com/problems/palindrome-partitioning)

## [分割回文串 II](https://leetcode-cn.com/problems/palindrome-partitioning-ii)

## [格雷编码](https://leetcode-cn.com/problems/gray-code)

class Solution {

public:

vector<int> grayCode(int n)

{

vector<int> res;

res.push\_back(0);

for(int i = 0;i < n;++i)//n位

{

for(int j = res.size()-1;j >= 0;--j)//每次从后往前遍历，能保障相邻的只有一个字符不同

{

res.push\_back((1 << i) + res[j]);

}

}

return res;

}

};

## [解码方法](https://leetcode-cn.com/problems/decode-ways)

## [买卖股票的最佳时机](https://leetcode-cn.com/problems/best-time-to-buy-and-sell-stock)

class Solution {

public:

    int maxProfit(vector<int>& prices) {

        int res(0),loc(0);

        int mi(INT\_MAX);

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

        {

            mi = min(prices[i],mi);

            loc = max(loc,prices[i]-mi);

            res = max(res,loc);

        }

        return res;

    }

};

## [买卖股票的最佳时机 II](https://leetcode-cn.com/problems/best-time-to-buy-and-sell-stock-ii)

class Solution {

public:

    int maxProfit(vector<int>& prices) {

        int res(0);

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

        {

            res += max(prices[i] - prices[i-1],0);

        }

        return res;

    }

};

## [买卖股票的最佳时机 III](https://leetcode-cn.com/problems/best-time-to-buy-and-sell-stock-iii)

class Solution {

public:

    int maxProfit(vector<int>& prices) {

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

        int g[3] = {0};//g[j]为最多可进行j次交易的最大利润，此为全局最优

        int l[3] = {0};//l[j]为最多可进行j次交易并且最后一次交易在最后一天卖出的最大利润，此为局部最优

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

            int diff = prices[i] - prices[i-1];

            for (int j = 2; j >= 1; --j)

            {

                l[j] = max(g[j - 1] + max(diff, 0), l[j] + diff);//需要从后往前计算，否则影响后面的计算

                g[j] = max(l[j], g[j]);

            }

        }

        return g[2];

    }

    /\*

local[j] = max(global[j - 1] + max(diff, 0), local[j] + diff) //局部最优值是比较前一天并少交易一次的全局最优加上大于0的差值，和前一天的局部最优加上差值中取较大值

global[j] = max(local[j], global[j]) //全局最优比较局部最优和前一天的全局最优

    \*/

};

## [编辑距离](https://leetcode-cn.com/problems/edit-distance)

class Solution {

public:

    int minDistance(string word1, string word2) {

        if (word1.size() == 0 && word2.size() == 0)return 0;

        int size1 = word1.size(), size2 = word2.size();

        vector<vector<int>> dp(size1 + 1, vector<int>(size2 + 1, 0));

        for (int i = 0; i <= size1; i++) dp[i][0] = i;//边际条件

        for (int j = 0; j <= size2; j++) dp[0][j] = j;

        for (int i = 1; i <= size1; i++) {

            for (int j = 1; j <= size2; j++) {

                int replace = word1[i - 1] == word2[j - 1] ? dp[i - 1][j - 1] : dp[i - 1][j - 1] + 1;//替换或相同

                int ins\_del = min(dp[i][j - 1] + 1, dp[i - 1][j] + 1);//插入或者删除

                dp[i][j] = min(replace, ins\_del);

            }

        }

        return dp[size1][size2];

    }

};

## [不同路径](https://leetcode-cn.com/problems/unique-paths)

class Solution {

public:

    int uniquePaths(int m, int n) {

        vector<vector<int>> array(m,vector<int>(n,1));

        for(int i=1; i<m;i++)//直接从第二行、第二列开始计算

        {

            for(int j=1;j<n;j++)

            {

                array[i][j]=array[i-1][j]+array[i][j-1];  //dp,每个格子来源于左边和上边的格子之和

            }

        }

        return array[m-1][n-1];

    }

};

## [不同路径 II](https://leetcode-cn.com/problems/unique-paths-ii)

class Solution {

public:

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

        if (!obstacleGrid.size() || !obstacleGrid[0].size())return 0;

        int m = obstacleGrid.size();

        int n = obstacleGrid[0].size();

        vector<vector<long long>> array(m,vector<long long>(n,0));

        for(int i = 0;i < m && !obstacleGrid[i][0];++i)array[i][0] = 1;

        for(int i = 0;i < n && !obstacleGrid[0][i];++i)array[0][i] = 1;

        for(int i=1; i<m;i++){

            for(int j=1;j<n;j++){

                if (!obstacleGrid[i][j])

                array[i][j]=array[i-1][j]+array[i][j-1];

            }

        }

        return array[m-1][n-1];

    }

};

## [最小路径和](https://leetcode-cn.com/problems/minimum-path-sum)

class Solution {

public:

    int minPathSum(vector<vector<int>>& grid) {

        int m = grid.size();

        int n = grid[0].size();

        int paths[m][n];

        paths[0][0] = grid[0][0];

        for(int i = 1; i < m; ++i) paths[i][0] = paths[i - 1][0] + grid[i][0];

        for(int j = 1; j < n; ++j)  paths[0][j] = paths[0][j - 1] + grid[0][j];

        for(int i = 1; i < m; ++i)

        for(int j = 1; j < n; ++j)

        paths[i][j] = std::min(paths[i - 1][j], paths[i][j - 1]) + grid[i][j];

        return paths[m - 1][n - 1];

    }

};

## [跳跃游戏](https://leetcode-cn.com/problems/jump-game)

class Solution {

public:

    bool canJump(vector<int>& nums) {

          int rightMost = 0;

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

            if (rightMost < i) break;

            rightMost = max(rightMost, i + nums[i]);

          }

          return rightMost >= (nums.size() - 1);

    }

};

## [跳跃游戏 II](https://leetcode-cn.com/problems/jump-game-ii)

class Solution {

public:

    int jump(vector<int>& nums) {

        if (nums.size() <= 1) return 0;

        int curPos = 0,lastPos = 0,cnt = 0;

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

        {

            curPos = max(nums[i]+i,curPos);//最大距离

            if (i == lastPos)//超过位置才能计数

            {

                lastPos = curPos;

                cnt++;

                if (curPos>=nums.size()-1)break;//已到

            }

        }

        return cnt;

    }

};

## [最大整除子集](https://leetcode-cn.com/problems/largest-divisible-subset)

## [H指数](https://leetcode-cn.com/problems/h-index)

坐标和数值的关系，利用了有序性

class Solution {

public:

    int hIndex(vector<int>& citations)

    {

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

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

        int h(0);

        for(int i = citations.size() -1;i >= 0;i--)

        {

            if (citations.size() - i <= citations[i])

            {

                ++h;

            }

            else break;

        }

        return h;

    }

};

## [H指数 II](https://leetcode-cn.com/problems/h-index-ii)

坐标和数值的关系，利用了有序性

class Solution {

public:

    int hIndex(vector<int>& citations)

    {

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

        int h(0);

        for(int i = citations.size() -1;i >= 0;i--)

        {

            if (citations.size() - i <= citations[i])

            {

                h = max((int)citations.size() - i,h);

            }

        }

        return h;

    }

};

[零钱兑换](https://leetcode-cn.com/problems/coin-change)

第k个与前面某个的关系

class Solution {

public:

    int coinChange(vector<int>& coins, int amount)

    {

        if (!amount)return 0;

        if (coins.size() == 0)return -1;

        long dp[amount+1] = {0};

        for(long i = 1;i <= amount;++i)

        {

            for(auto& c:coins)

            {

                int j = i - c;

                if (j>= 0 && (j == 0||dp[j]))

                {

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

                }

            }

        }

        return dp[amount] > 0? dp[amount]:-1;

    }

};

## [打家劫舍](https://leetcode-cn.com/problems/house-robber)

class Solution {

public:

    int rob(vector<int>& nums) {

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

        int v1(0),v2 = nums[0],l,g;

        g = v2;

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

        {

            l = max(v1 + nums[i],v2);//局部最优

            v1 = v2;v2 = l;//局部推移

            g = max(g,l);//全局最优

        }

        return g;

    }

};

## [打家劫舍 II](https://leetcode-cn.com/problems/house-robber-ii)

第一个开始，倒数2结束；或者第二个开始，倒数一结束

class Solution {

public:

    int rob(vector<int>& nums) {

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

        if (nums.size() == 1)return nums[0];

        return max(rob(nums,0,nums.size()-1),rob(nums,1,nums.size()));

    }

    int rob(vector<int>& nums,int b,int e)//[b,e)

    {

        int size = e - b;

        if(size == 0) return 0;

        int v1 = 0,v2 = nums[b],l,g = nums[b];

        for(int i = b + 1;i < e;++i)

        {

            l = max(nums[i] + v1,v2);//局部最优

            v1 = v2;

            v2 = l;//局部推移

            g = max(l,g);//全局最优

        }

        return g;

    }

};

## [打家劫舍 III](https://leetcode-cn.com/problems/house-robber-iii)

dp。每个子树含取root，不取root两个状态的值，也是局部最优解；父树依赖子树状态来得到自身两状态，最终是全局最优解。

class Solution {

public:

    vector<int> getMoney(TreeNode\* node) {

        vector<int> ret(2, 0);

        if(!node) return ret;

        vector<int> lRet = getMoney(node->left);

        vector<int> rRet = getMoney(node->right);

        ret[0] = lRet[1] + rRet[1] + node->val;  //取root

        ret[1] = max(lRet[0], lRet[1]) + max(rRet[0], rRet[1]);//不取root

        return ret;

    }

    int rob(TreeNode\* root) {

        vector<int> ret = getMoney(root);

        return max(ret[0], ret[1]);

    }

};

## [最长上升子序列](https://leetcode-cn.com/problems/longest-increasing-subsequence)

## [鸡蛋掉落](https://leetcode-cn.com/problems/super-egg-drop)

class Solution {

public:

    int superEggDrop(int K, int N) {

        vector<int> dp(K + 1, 0);

        int m(0);

        for (;dp[K] < N;m++)

        {

            for (int k = K; k > 0; --k)//从上往下计算

                dp[k] += dp[k - 1] + 1;//分成三段 上 、下 、本层

        }

        return m;

    }

};

# 集合

## [天际线问题](https://leetcode-cn.com/problems/the-skyline-problem)

## [常数时间插入、删除和获取随机元素](https://leetcode-cn.com/problems/insert-delete-getrandom-o1)

# 几何

## [矩形面积](https://leetcode-cn.com/problems/rectangle-area)

## [直线上最多的点数](https://leetcode-cn.com/problems/max-points-on-a-line)

# 拓扑结构

## [课程表](https://leetcode-cn.com/problems/course-schedule)

利用了哈希表记录入度表，利用哈希表记录入度的出度，利用set来遍历

class Solution {

public:

    bool canFinish(int numCourses, vector<pair<int, int>>& prerequisites) {

        set<int> st;

        unordered\_map<int,int> m;// 入度的数量

        unordered\_map<int,vector<int>> dep;//出度的所有入度（依赖关系）

        for(auto& iter:prerequisites)

        {

            m[iter.first]++;//入度计数

            dep[iter.second].push\_back(iter.first);

            st.insert(iter.first);//只是处理有依赖的课程

            st.insert(iter.second);

        }

        while(st.size())

        {

            int left = st.size();

            for(auto iter = st.begin();iter!=st.end();)

            {

                int learn = \*iter;

                if (m[learn] == 0)//没有入度的

                {

                    //printf("1)%d\n",learn);

                    for(auto i:dep[learn])//减少所有出度的入度（减少依赖）

                    {

                        if (m[i])m[i]--;

                    }

                    st.erase(iter++);

                }

                else

                {

                    iter++;

                }

            }

            if (left == st.size())return false;//没有减少课程的则说明不能继续

        }

        return true;

    }

};

## [课程表 II](https://leetcode-cn.com/problems/course-schedule-ii)

因为只需要返回一种，可以直接遍历

class Solution {

public:

vector<int> findOrder(int numCourses, vector<pair<int, int>>& prerequisites) {

unordered\_map<int,int> inM;//入度列表

set<int> courseSt;//课程

unordered\_map<int,vector<int>> depM;//出度列表

for(auto &iter:prerequisites)

{

inM[iter.first]++;

depM[iter.second].push\_back(iter.first);

courseSt.insert(iter.first);

courseSt.insert(iter.second);

}

vector<int> res;

for(int i = 0;i < numCourses;++i)

{

//printf("1)%d\n",i);

if (!courseSt.count(i))res.push\_back(i);//没有依赖的则先学习

}

while(courseSt.size())

{

int left = courseSt.size();

for(auto iter = courseSt.begin();iter != courseSt.end();)

{

if (inM[\*iter] == 0)

{

//printf("2)%d\n",\*iter);

res.push\_back(\*iter);

for(auto i:depM[\*iter])

{

if (inM[i]) inM[i]--;

}

courseSt.erase(iter++);

}

else

{

iter++;

}

}

if (courseSt.size() == left)return {};//不能完成

}

return res;

}

};

## [单词接龙 II](https://leetcode-cn.com/problems/word-ladder-ii)

拓扑图的方式一层层遍历获取临接节点，临接节点为修改一个字符就出现在字典内的；每遍历一层后再从字典删除该前一层的节点。每一个新的节点、新的路径，需要拷贝，因为需要发现所有的可能的路径（所有的有向无环图）。

class Solution {

public:

    vector<vector<string>> findLadders(string beginWord, string endWord, vector<string>& wordList)

    {

        vector<vector<string>> res;

        unordered\_set<string> dict(wordList.begin(), wordList.end());

        vector<string> path{beginWord};

        queue<vector<string>> paths;

        paths.push(path);

        int level = 1, minLevel = INT\_MAX;

        unordered\_set<string> words;

        while (paths.size()) {

            auto path = paths.front(); paths.pop();//每个path都处理

            if (path.size() > level) {//到了新的一层

                for (string w : words) dict.erase(w);//处理掉前一层的词，否则会回环

                words.clear();

                level = path.size();

                if (level > minLevel) break;//长度已超过最短长度，则表示已到最后一层

            }

            string last = path.back();

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

                string newLast = last;

                for (char ch = 'a'; ch <= 'z'; ++ch) {

                    newLast[i] = ch;

                    if (dict.count(newLast))

                    {

                        words.insert(newLast);//本层的单词

                        vector<string> nextPath = path;//新的路径的尝试

                        nextPath.push\_back(newLast);

                        if (newLast == endWord) {

                            res.push\_back(nextPath);

                            minLevel = level;

                        }

                        else paths.push(nextPath);

                    }

                }

            }

        }

        return res;

    }

};

## [单词接龙](https://leetcode-cn.com/problems/word-ladder)

拓扑图的方式一层层遍历获取临接节点，临接节点为修改一个字符就出现在字典内的；每遍历一个临接节点就从字典删除该节点

class Solution {

public:

    int ladderLength(string beginWord, string endWord, vector<string>& wordList)

    {

        if(beginWord.size() == 0|| endWord.size() == 0||beginWord.size() != endWord.size() || wordList.size() == 0)return 0;

        //拓扑图的思想

        unordered\_set<string> ws(wordList.begin(),wordList.end());

        unordered\_map<string,int> wl;//临接网

        list<string> l;//遍历的队列

        l.push\_back(beginWord);

        wl[beginWord] = 1;

        while(l.size())

        {

            string w = l.front();

            l.pop\_front();

            int level = wl[w];

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

            {

                char c = w[i];

                for(int j = 'a';j <= 'z';++j)

                {

                    if (w[i] == (char)j)continue;

                    w[i] = (char)j;

                    if (ws.count(w) > 0)//测试所有的可能的临接节点

                    {

                        //printf("%s %d\n",w.c\_str(),level);

                        if (w == endWord) return level + 1;

                        ws.erase(w);

                        l.push\_back(w);

                        wl[w] = level + 1;

                    }

                }

                w[i] = c;

            }

        }

        return 0;

    }

};

# 前缀树

## [实现 Trie (前缀树)](https://leetcode-cn.com/problems/implement-trie-prefix-tree)

本质上是多叉树，利用了当前的访问的状态能够依赖之前访问的状态

class Trie {

public:

    struct node

    {

        node():isWord(false){memset(children,0,sizeof(children));}

        ~node(){for(int i = 0;i < 26;++i){if (children[i]) {delete children[i];children[i] = NULL; } }}

        node\* children[26];

        bool isWord;

    };

    node \*root;

    /\*\* Initialize your data structure here. \*/

    Trie() {

        root = new node();

    }

    ~Trie(){delete root;}

    /\*\* Inserts a word into the trie. \*/

    void insert(string word) {

        if (word.size() == 0)return;

        node \*p = root;

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

        {

            if (!p->children[word[i] - 'a']) p->children[word[i] - 'a'] = new node();

            p = p->children[word[i] - 'a'];

        }

        p->isWord = true;

    }

    /\*\* Returns if the word is in the trie. \*/

    bool search(string word)

    {

        if (word.size() == 0)return true;

        node \*p = root;

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

        {

            if (!p->children[word[i] - 'a']) return false;

            p = p->children[word[i] - 'a'];

        }

        return p->isWord;

    }

    /\*\* Returns if there is any word in the trie that starts with the given prefix. \*/

    bool startsWith(string prefix) {

        if (prefix.size() == 0)return true;

        node \*p = root;

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

        {

            if (!p->children[prefix[i] - 'a']) return false;

            p = p->children[prefix[i] - 'a'];

        }

        return true;

    }

};

## [添加与搜索单词 - 数据结构设计](https://leetcode-cn.com/problems/add-and-search-word-data-structure-design)

class WordDictionary {

public:

struct TrieNode {

public:

TrieNode \*child[26];

bool isWord;

TrieNode() : isWord(false)

{

memset(child,0,sizeof(child));

}

};

TrieNode \*root;

WordDictionary() {

root = new TrieNode();

}

~WordDictionary(){delete root;}

// Adds a word into the data structure.

void addWord(const string &word) {

TrieNode \*p = root;

for (auto &i : word) {

if (!p->child[i - 'a']) p->child[i - 'a'] = new TrieNode();

p = p->child[i - 'a'];

}

p->isWord = true;

}

// Returns if the word is in the data structure. A word could

// contain the dot character '.' to represent any one letter.

bool search(const string &word) {

if (word.size() == 0)return false;

return dfs(word, root, 0);

}

bool dfs(const string &word, TrieNode \*p, int pos) {

if (pos == word.size()) return p->isWord;

for(int i = pos;i < word.size();++i)

{

if (word[i] == '.')

{

for (auto &child : p->child) {

if (child && dfs(word, child, i + 1)) return true;

}

return false;

}

if (!p->child[word[i] - 'a'])return false;

p = p->child[word[i] - 'a'];

}

return p->isWord;

}

};

/\*

这道题如果做过之前的那道 Implement Trie (Prefix Tree) 实现字典树(前缀树)的话就没有太大的难度了，还是要用到字典树的结构，唯一不同的地方就是search的函数需要重新写一下，因为这道题里面'.'可以代替任意字符，所以一旦有了'.'，就需要查找所有的子树，只要有一个返回true，整个search函数就返回true，典型的DFS的问题，其他部分跟上一道实现字典树没有太大区别

\*/

/\*\*

\* Your WordDictionary object will be instantiated and called as such:

\* WordDictionary obj = new WordDictionary();

\* obj.addWord(word);

\* bool param\_2 = obj.search(word);

\*/

# 回溯

## [被围绕的区域](https://leetcode-cn.com/problems/surrounded-regions)

## [单词搜索](https://leetcode-cn.com/problems/word-search)

回溯就是使用多个方向的dfs来搜索，过程中为了避免回环需要设置路径成员为特殊字符。

class Solution {

public:

    int height;int width;

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

        height=board.size();

        if(height==0)  return false;

        width=board[0].size();

        for(int i=0;i<height;i++)

            for(int j=0;j<width;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 pos)//从节点board[i][j]开始查找word

    {

        if(i<0||j<0||i>=height||j>=width||board[i][j]=='\0'||board[i][j]!=word[pos])return false;

        if(pos==word.size()-1)return true;

        char t=board[i][j];

        board[i][j]='\0';

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

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

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

           dfs(board,word,i,j-1,pos+1))  return true;

        board[i][j]=t;

        return false;

    }

};

## [单词搜索 II](https://leetcode-cn.com/problems/word-search-ii)

class Solution {

public:

    int height;int width;

    vector<string> findWords(vector<vector<char>>& board, vector<string>& words) {

        vector<string> res;

        unordered\_set<string> setRes;

        height=board.size();

        if(height==0)  return res;

        width=board[0].size();

        if (words.size() == 0)return res;

        for(const auto &w :words)

        {

            if (exist(board,w))setRes.insert(w);

        }

        for(auto &s:setRes)res.emplace\_back(s);

        return res;

    }

    //find

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

        for(int i=0;i<height;i++)

            for(int j=0;j<width;j++)

               if(backtrack(board,word,i,j,0)) return true;

        return false;

    }

    bool backtrack(vector<vector<char>>& board,const string& word,int i,int j,int pos)//从节点board[i][j]开始查找word

    {

        if(i<0||j<0||i>=height||j>=width||board[i][j]==0||board[i][j]!=word[pos])return false;

        if(pos==word.size()-1)return true;

        char t=board[i][j];

        board[i][j]=0;

        if(backtrack(board,word,i,j+1,pos+1)||

           backtrack(board,word,i+1,j,pos+1)||

           backtrack(board,word,i-1,j,pos+1)||

           backtrack(board,word,i,j-1,pos+1))

        {

            board[i][j]=t;

            return true;

        }

        board[i][j]=t;

        return false;

    }

};

## [矩阵置零](https://leetcode-cn.com/problems/set-matrix-zeroes)

## [矩阵中的最长递增路径](https://leetcode-cn.com/problems/longest-increasing-path-in-a-matrix)

## [岛屿的最大面积](https://leetcode-cn.com/problems/max-area-of-island)

## [岛屿的个数](https://leetcode-cn.com/problems/number-of-islands)

回溯就是使用多个方向的dfs来搜索，过程中为了避免回环需要设置路径成员为特殊字符。找到一个岛屿计数一次

class Solution {

public:

    int m,n,count = 0;

    int numIslands(vector<vector<char>>& grid) {

        if(grid.size() == 0 || grid[0].size() == 0)return 0;

        m = grid.size();

        n = grid[0].size();

        for(int i = 0;i < m;++i)

        {

            for(int j = 0;j < n;++j)

            {

                if (grid[i][j] == '1')

                {

                    ++count;

                    extend(grid,i,j);

                }

            }

        }

        return count;

    }

    void extend(vector<vector<char>>& grid,int i,int j)

    {

        if (i >= 0 && i < m && j >= 0 && j < n)

        {

            if (grid[i][j] == '1')

            {

                grid[i][j] = '2';

                extend(grid,i,j+1);

                extend(grid,i,j-1);

                extend(grid,i+1,j);

                extend(grid,i-1,j);

            }

        }

    }

};

# 位操作

[2的幂](https://leetcode-cn.com/problems/power-of-two)

class Solution {

public:

//n & (n-1) 就是为了 去掉最右边的位数 1.剩下的为0则为2 的幂

bool isPowerOfTwo(int n) {

return (n>0) && !(n & (n-1));

}

};

[4的幂](https://leetcode-cn.com/problems/power-of-four)

因为整形是4字节的，再利用1位的特性。

class Solution {

public:

    bool isPowerOfFour(int num) {

        if (num <= 0)return false;

        if(num & (num - 1)) return false; // 多个位

        return  (0x55555555 & num) > 0;

    }

};

## [位1的个数](https://leetcode-cn.com/problems/number-of-1-bits)

每次去掉最后一位

class Solution {

public:

    int hammingWeight(uint32\_t n) {

        int c(0);

        while(n)

        {

            n = n & (n-1);

            ++c;

        }

        return c;

    }

};

## [只出现一次的数字](https://leetcode-cn.com/problems/single-number)

异或两相同数会抵消，剩下的就是一次的

class Solution {

public:

    int singleNumber(vector<int>& nums) {

        int res(0);

        for(auto &n:nums)

        {

            res ^= n;

        }

        return res;

    }

};

## [只出现一次的数字 II](https://leetcode-cn.com/problems/single-number-ii)

遍历，记录出现一次的位，出现2次的位为之前出现一次且本数，算了2次后才能算3次，3次的为出现1次且2次

class Solution {

public:

    int singleNumber(vector<int>& nums) {

        int ones = 0;//记录只出现过1次的bits

        int twos = 0;//记录只出现过2次的bits

        int threes;

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

            int t = nums[i];

            twos |= ones&t;//要在更新ones前面更新twos

            ones ^= t;

            threes = ones&twos;//ones和twos中都为1即出现了3次

            ones &= ~threes;//抹去出现了3次的bits

            twos &= ~threes;

        }

        return ones;

    }

};

## [只出现一次的数字 III](https://leetcode-cn.com/problems/single-number-iii)

异或操作让出现两次的会抵消了，剩下的两个数发异或，与其自身补码与操作，则是获取某数中的一位，该位只存在于该数中；再重新遍历判断和异或所有数

class Solution {

public:

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

        int AXORB = 0;

        for (int num : nums) {

            AXORB ^= num;

        }

        int bitFlag = AXORB & (~AXORB + 1);//与自身的补码是为了获取其中一位,并且该位只是存在于其中一个数中

        vector<int> res = vector<int>(2,0);

        for (int num : nums) {

            if ((num & bitFlag) == 0) {

                res[0] ^= num;

            } else {

                res[1] ^= num;

            }

        }

        return res;

    }

};

# 常用结构

# 链表

（缓存）

## [LRU缓存机制](https://leetcode-cn.com/problems/lru-cache)

利用哈希表的快速访问，记录的是链表迭代器，因为迭代器需要被移动；链表的访问顺序判断访问热度

class LRUCache {

public:

    LRUCache(int capacity)

    {

        s = capacity;

    }

    int get(int key)

    {

        auto it = m.find(key);

        if (it != m.end())

        {

            l.splice(l.begin(),l,it->second);

            return it->second->second;

        }

        return -1;

    }

    void put(int key, int value)

    {

        auto it = m.find(key);

        if (it != m.end())

        {

            l.splice(l.begin(),l,it->second);

            it->second->second = value;

        }

        else

        {

            if (l.size() + 1> s)

            {

                m.erase(l.back().first);

                l.pop\_back();

            }

            l.push\_front(make\_pair(key,value));

            m[key] = l.begin();

        }

    }

private:

    unordered\_map<int,list<pair<int,int>>::iterator> m;//key value (iter)

    list<pair<int,int>> l;//key value

    int s;

};

## [LFU缓存](https://leetcode-cn.com/problems/lfu-cache)

lfu（最不经常使用的淘汰掉算法）可以处理缓存污染问题（是指系统将不常用的数据从内存移到缓存，造成常用数据的挤出，降低了缓存效率的现象）

需要记录最少使用次数，需要用哈希表记录次数对应的键列表，再用哈希表记录键、值和次数，还有哈希表记录键在次数列表中的迭代器。

每次获取键则需要更新次数对应的键列表，和其键对应的迭代器（因为所在的列表变了）。

每次插入则先获取，有就更新值，没则可能需要移除旧的次数最小的和最少使用的键，然后插入新值，新值次数都为1.

class LFUCache {

public:

    LFUCache(int capacity) {

        cap = capacity;

        minfre = 0;

    }

    int get(int key)

    {

        if (keyVFre.count(key) == 0) return -1;

        //处理键的使用频次

        int& kfre = keyVFre[key].second;//键的使用次数，修改次数，记录键的使用次数迭代器

        freKeys[kfre].erase(keyFreIter[key]);//删除次数列表中的键的迭代器（为了效率才记录）

        freKeys[++kfre].push\_back(key);

        keyFreIter[key] = --freKeys[kfre].end();//不能没有记录键的次数的列表的跌代器，为了管理键的频次

        if (freKeys[minfre].size() == 0) ++minfre;//判断最小次数没有了就上升，因为是访问。贪心算法方式

        return keyVFre[key].first;

    }

    void put(int key, int value) {

        if (cap <= 0) return;

        if (get(key) != -1) {

            keyVFre[key].first = value;

            return;

        }

        if (keyVFre.size() >= cap)

        {//erase  fre

            auto& l = freKeys[minfre];

            keyVFre.erase(l.front());

            keyFreIter.erase(l.front());

            l.pop\_front();

            //printf("minfre %d\n",minfre);

        }

        {//new one

            keyVFre[key] = {value, 1};

            freKeys[1].push\_back(key);

            keyFreIter[key] = --freKeys[1].end();

            minfre = 1;

        }

    }

    int cap, minfre;

    unordered\_map<int, pair<int, int>> keyVFre;// key : val fre

    unordered\_map<int, list<int>> freKeys;// fre : keys

    unordered\_map<int, list<int>::iterator> keyFreIter;// key : freKeys iter

};

（其他）

## [复制带随机指针的链表](https://leetcode-cn.com/problems/copy-list-with-random-pointer)

/\*

// Definition for a Node.

class Node {

public:

int val;

Node\* next;

Node\* random;

Node() {}

Node(int \_val, Node\* \_next, Node\* \_random) {

val = \_val;

next = \_next;

random = \_random;

}

};

\*/

class Solution {

public:

Node\* dummy;

Solution(){dummy = new Node();}

~Solution(){delete dummy;}

Node\* copyRandomList(Node\* head) {

if (!head)return NULL;

unordered\_map<Node\*,Node\*> mp;//old => new

for(Node\* tmp = head,\* tmpDummy = dummy;tmp;tmp = tmp->next)

{

tmpDummy->next = mp[tmp] = new Node(tmp->val,NULL,NULL);//先处理next指针，哈希表中节点映射的是新链表的节点

tmpDummy = tmpDummy->next;

}

for(Node\* tmp = head;tmp;tmp = tmp->next)

{

mp[tmp]->random = mp[tmp->random];//再处理random指针，哈希表中的random映射的是新链表的节点

}

return dummy->next;

}

};

## [删除链表中的节点](https://leetcode-cn.com/problems/delete-node-in-a-linked-list)

## [相交链表](https://leetcode-cn.com/problems/intersection-of-two-linked-lists)

## [对链表进行插入排序](https://leetcode-cn.com/problems/insertion-sort-list)

## [排序链表](https://leetcode-cn.com/problems/sort-list)

## [对链表进行插入排序](https://leetcode-cn.com/problems/insertion-sort-list)

## [环形链表](https://leetcode-cn.com/problems/linked-list-cycle)

使用快慢指针的方式遍历判断

class Solution {

public:

    bool hasCycle(ListNode \*head)

    {

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

        while(fast && fast->next)

        {

            if (slow == fast->next || slow == fast->next->next)return true;

            slow = slow->next,fast = fast->next->next;

        }

        return false;

    }

};

## [环形链表 II](https://leetcode-cn.com/problems/linked-list-cycle-ii)

使用快慢指针找到重叠，重置慢指针，再找到重叠就是入环点

class Solution {

public:

    ListNode \*detectCycle(ListNode \*head) {

        if( head == NULL || head->next == NULL )return NULL;

        ListNode\* fp = head,\* sp = head;

        while( fp  && fp->next )

        {

            sp = sp->next;

            fp = fp->next->next;

            if( fp == sp )

            {

                break;

            }

        }

        if( !fp || !fp->next )return NULL;

        sp = head;

        while( fp != sp )

        {

            sp = sp->next;

            fp = fp->next;

        }

        return sp;

    }

};

## [重排链表](https://leetcode-cn.com/problems/reorder-list)

## [删除排序链表中的重复元素](https://leetcode-cn.com/problems/remove-duplicates-from-sorted-list)

## [删除排序链表中的重复元素 II](https://leetcode-cn.com/problems/remove-duplicates-from-sorted-list-ii)

## [分隔链表](https://leetcode-cn.com/problems/partition-list)

## [旋转链表](https://leetcode-cn.com/problems/rotate-list)

连成一圈再断开，注意计算断开位置

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

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

\* };

\*/

class Solution {

public:

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

if (!head || !head->next) return head;

// 获取长度和末尾节点tail

int len = 1;

ListNode\* tail = head;

while (tail->next && len++) tail = tail->next;

if (k % len == 0) return head;

// 找到倒数第k%len(也是断开节点的)的前一个节点

int pos = len - k % len;

ListNode \*pre = head;

while (--pos) pre = pre->next;

// 连成一个圈

tail->next = head;

//从pre位置处断开

head = pre->next;

pre->next = NULL;

return head;

}

};

## [移除链表元素](https://leetcode-cn.com/problems/remove-linked-list-elements)

## [反转链表](https://leetcode-cn.com/problems/reverse-linked-list)

## [反转链表 II](https://leetcode-cn.com/problems/reverse-linked-list-ii)

## [合并两个有序链表](https://leetcode-cn.com/problems/merge-two-sorted-lists)

## [合并K个排序链表](https://leetcode-cn.com/problems/merge-k-sorted-lists)

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

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

\* };

\*/

class Solution {

public:

ListNode \* dummy;

Solution(){dummy = new ListNode(-1);}

~Solution(){delete dummy;}

ListNode\* mergeKLists(vector<ListNode\*>& lists) {

if(lists.empty()){

return nullptr;

}

list<ListNode\*> l;

for(auto h:lists)l.emplace\_back(h);

while(l.size() > 1){

ListNode\* l1 = l.front();

l.pop\_front();

ListNode\* l2 = l.front();

l.pop\_front();

l.emplace\_back(mergeTwoLists(l1,l2));

}

return l.front();

}

ListNode \*mergeTwoLists(ListNode \*l1, ListNode \*l2) {

if(!l1 || !l2) return l1? l1:l2;

ListNode \* tmp = dummy;

while (l1 && l2)

{

if(l1->val <= l2->val)

{

tmp ->next = l1;

l1 = l1->next;

}

else

{

tmp->next = l2;

l2 = l2->next;

}

tmp = tmp->next;

}

if (l1) tmp->next = l1;

else tmp->next = l2;

return dummy->next;

}

/\*

考虑分治的思想来解这个题（类似归并排序的思路）。把这些链表分成两半，如果每一半都合并好了，那么我就最后把这两个合并了就行了。这就是分治法的核心思想。

但是这道题由于存的都是指针，就具有了更大的操作灵活性，可以不用递归来实现分治。就是先两两合并后在两两合并。。。一直下去直到最后成了一个。（相当于分治算法的那棵二叉树从底向上走了）。

第一次两两合并是进行了k/2次，每次处理2n个值,即2n \* k/2 = kn 次比较。

第二次两两合并是进行了k/4次，每次处理4n个值,即4n \* k/4 = kn 次比较。

。。。

最后一次两两合并是进行了k/(2^logk)次（=1次），每次处理2^logK \* N个值（kn个），即1\*kn= kn 次比较。

所以时间复杂度：

O(KN\* logK)

空间复杂度是O(1)。

\*/

};

## [两两交换链表中的节点](https://leetcode-cn.com/problems/swap-nodes-in-pairs)

## [k个一组翻转链表](https://leetcode-cn.com/problems/reverse-nodes-in-k-group)

## [删除链表的倒数第N个节点](https://leetcode-cn.com/problems/remove-nth-node-from-end-of-list)

## [回文链表](https://leetcode-cn.com/problems/palindrome-linked-list)

## [奇偶链表](https://leetcode-cn.com/problems/odd-even-linked-list)

## [两数相加](https://leetcode-cn.com/problems/add-two-numbers)

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

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

\* };

\*/

class Solution {

public:

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

{

if (!l1 && !l2)return NULL;

ListNode\* head = new ListNode(0);

ListNode\* tmp = head;

int val(0);

while (l1 || l2 || val)//由于链表本身也是从低到高位的，可以直接从头计算

{

val = (l1 ? l1->val:0 ) + (l2 ? l2->val:0) + val;

head->next = new ListNode(val % 10);

val /= 10;//进位

if (l1)l1 = l1->next;

if (l2)l2 = l2->next;

head = head->next;

}

head = tmp->next;

delete tmp;

return head;

}

};

## [两数相加 II](https://leetcode-cn.com/problems/add-two-numbers-ii)

# 栈

## [用队列实现栈](https://leetcode-cn.com/problems/implement-stack-using-queues)

## [用栈实现队列](https://leetcode-cn.com/problems/implement-queue-using-stacks)

## [最小栈](https://leetcode-cn.com/problems/min-stack)

class MinStack {

public:

/\*\* initialize your data structure here. \*/

MinStack() {

}

void push(int x) {

s.push(x);

if (sMin.size() == 0 || sMin.top() >= x)sMin.push(x);

}

void pop() {

int x = s.top();

s.pop();

if (x == sMin.top()) sMin.pop();

}

int top() {

return s.top();

}

int getMin() {

return sMin.top();

}

stack<int> s;

stack<int> sMin;

};

## [有效的括号](https://leetcode-cn.com/problems/valid-parentheses)

class Solution {

public:

    bool isValid(string s)

    {

        int top=-1,index=0,length=s.size();

        string stack;

        stack.resize(length,0);

        while(index < length)

        {

            if(s[index]==')')//弹栈操作

            {

                if(top>=0 && stack[top]=='(')top--;

                else return false;

            }

            else if(s[index]=='}')

            {

                if(top>=0 && stack[top]=='{')top--;

                else return false;

            }

            else if(s[index]==']')

            {

                if(top>=0 && stack[top]=='[')top--;

                else return false;

            }

            else stack[++top]=s[index];//压栈操作

            index++;

        }

        return top==-1;

    }

};

## [下一个更大元素 I](https://leetcode-cn.com/problems/next-greater-element-i)

class Solution {

public:

    vector<int> nextGreaterElement(vector<int>& findNums, vector<int>& nums) {

        vector<int> res;

        stack<int> st;

        unordered\_map<int, int> m;

        for (int num : nums) {

            //和其右边第一个较大数之间的映射，因为没有重复元素就可以这样处理

            while (st.size() && st.top() < num) {

                m[st.top()] = num; st.pop();

            }

            st.push(num);//每个都要压

        }

        for (int num : findNums) {

            res.push\_back(m.count(num) ? m[num] : -1);

        }

        return res;

    }

};

## [下一个更大元素 II](https://leetcode-cn.com/problems/next-greater-element-ii)

class Solution {

public:

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

        int n = nums.size();

        vector<int> result(n, -1);

        stack<int> s;

        for (int i = 0; i < n; i++) {

            while (s.size() && nums[s.top()] < nums[i]) {//第一遍处理数在数组右边的第一个较大

                result[s.top()] = nums[i];

                s.pop();

            }

            s.push(i);//压下标，需要找回位置

        }

        for (int i = 0; i < n; i++) {

            while (s.size()&& nums[s.top()] < nums[i]) {//第二遍处理剩下的在循环数组右边的第一个较大

                result[s.top()] = nums[i];

                s.pop();

            }

        }

        return result;

    }

};

# 堆

例如，常用数据处理方式

最小堆用于处理前k大

最大堆用于处理前k小

priority\_queue<int,vector<int>,less<int>> pq;//最大堆，默认为priority\_queue<int>

priority\_queue<int,vector<int>,greater<int>> pq;//最小堆

struct cmp

{

        bool operator() (const pair<int, int> &a,const pair<int, int> &b)const {return a.second > b.second;} //b.second优先级高，也就是在堆上面

};

priority\_queue<pair<int,int>,vector<pair<int,int>>,cmp> pq;//最小堆

struct cmp{

        bool operator() (const pair<string, int> &a,const pair<string, int> &b)

        {

            if (a.second != b.second) return a.second > b.second;//b.second优先级高，也就是在堆上面

            return a.first < b.first;  //a.first优先级高，也就是在堆上面

        }

    };

priority\_queue<pair<string, int>, vector<pair<string, int>>,cmp> pq;  //次数的最小堆（字母的最大堆）

set 也可以当做唯一性的最小堆（因为红黑树的有序性，map同理）

## [数据流的中位数](https://leetcode-cn.com/problems/find-median-from-data-stream)

## [前K个高频元素](https://leetcode-cn.com/problems/top-k-frequent-elements)

class Solution {

public:

    struct cmp

    {

        bool operator() (const pair<int, int> &a,const pair<int, int> &b)const

        {

            return a.second > b.second;

        }

    };

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

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

        vector<int> res;

        unordered\_map<int,int> m;

        for(auto n:nums)m[n]++;

        priority\_queue<pair<int,int>,vector<pair<int,int>>,cmp> pq;//最小堆

        for(auto i:m)

        {

            pq.push(i);//使用pair成员比较容易兼容map的遍历成员

            if (pq.size() > k) pq.pop();

        }

        while(pq.size())

        {

            res.push\_back(pq.top().first);

            pq.pop();

        }

        return res;

    }

};

## [前K个高频单词](https://leetcode-cn.com/problems/top-k-frequent-words)

class Solution {

public:

    struct cmp{

        bool operator() (const pair<string, int> &a,const pair<string, int> &b)

        {

            if (a.second != b.second) return a.second > b.second;//b.second优先级高(次数最小堆)

            return a.first < b.first;  //a.first优先级高(字符串最大堆)

        }

    };

    vector<string> topKFrequent(vector<string>& words, int k) {

        unordered\_map<string, int> map;

        priority\_queue<pair<string, int>, vector<pair<string, int>>,cmp> pq;

        vector<string> res;

        for (string& s : words)

        {

            map[s]++;

        }

        for (auto& p : map)

        {

            pq.push(p);

            if (pq.size() > k)pq.pop();

        }

        while(pq.size() > 0)

        {

            res.push\_back(pq.top().first);

            pq.pop();

        }

        return res;

    }

};

## [最长有效括号](https://leetcode-cn.com/problems/longest-valid-parentheses)

# 数字

（找规律）

## [整数拆分](https://leetcode-cn.com/problems/integer-break)

## [Nim游戏](https://leetcode-cn.com/problems/nim-game)

## [加一](https://leetcode-cn.com/problems/plus-one)

# 其他

## [Excel表列序号](https://leetcode-cn.com/problems/excel-sheet-column-number)

[数字范围按位与](https://leetcode-cn.com/problems/bitwise-and-of-numbers-range)

[颠倒二进制位](https://leetcode-cn.com/problems/reverse-bits)

[分数到小数](https://leetcode-cn.com/problems/fraction-to-recurring-decimal)

[阶乘后的零](https://leetcode-cn.com/problems/factorial-trailing-zeroes)

[最大数](https://leetcode-cn.com/problems/largest-number)

[数字1的个数](https://leetcode-cn.com/problems/number-of-digit-one)

[移除元素](https://leetcode-cn.com/problems/remove-element)

[两数相除](https://leetcode-cn.com/problems/divide-two-integers)

[二进制求和](https://leetcode-cn.com/problems/add-binary)

## [x 的平方根](https://leetcode-cn.com/problems/sqrtx)

class Solution {

public:

    int mySqrt(int x) {

        double begin = 0; double end = x;

        double result = 1;  //需要从1开始，避免小数被取整时被清除

        double mid = 1;

        while(abs(x - result) > 0.000001){

            mid = (begin + end) / 2;

            result = mid  \* mid;

            if(result > x)  end = mid;

            else begin = mid;

        }

        return (int)mid;

    }

};

## [快乐数](https://leetcode-cn.com/problems/happy-number)

就是检查是否会循环

class Solution {

public:

    bool isHappy(int n)

    {

        unordered\_set<int> st;

        while(n != 1)

        {

            st.insert(n);

            int i(0);

            while (n)

            {

                i += (n % 10) \* (n % 10);

                n /=10;

            }

            if (st.count(i) > 0) return false;

            n = i;

        }

        return true;

    }

};

## [计数质数](https://leetcode-cn.com/problems/count-primes)

贪心方式填数

class Solution {

public:

    set<int> st;

    int countPrimes(int n) {

        if (n <=1)return 0;

        vector<bool> primes(n,true);

        for(int i = 2;i \* i < n;++i)

        {

            for(int j = i \* i;j < n;j += i)//i\*i起点，小的已被填

            {

                primes[j] = false;

            }

        }

        int c(0);

        for(int i = 2;i < n;++i)

        {

            if (primes[i]) ++c;

        }

        return c;

    }

};

## [整数替换](https://leetcode-cn.com/problems/integer-replacement)

## [求众数 II](https://leetcode-cn.com/problems/majority-element-ii)

## [整数转罗马数字](https://leetcode-cn.com/problems/integer-to-roman)

## [罗马数字转整数](https://leetcode-cn.com/problems/roman-to-integer)

## [完全平方数](https://leetcode-cn.com/problems/perfect-squares)

class Solution {

public:

int numSquares(int n)

{

if (n <= 1)return n;

int dp[n+1] = {0};

for (int i = 1;i <= n;++i)//从前往后动规

{

dp[i] = i;//最多的次数

for(int k = 1;k \* k <= i;++k)//所有可能的递推情况，选取次数少的

{

dp[i] = min(dp[i - k \* k] + 1,dp[i]);

}

}

return dp[n];

}

};

## [超级丑数](https://leetcode-cn.com/problems/super-ugly-number)

## [电话号码的字母组合](https://leetcode-cn.com/problems/letter-combinations-of-a-phone-number)

## [整数反转](https://leetcode-cn.com/problems/reverse-integer)

## [字符串转换整数 (atoi)](https://leetcode-cn.com/problems/string-to-integer-atoi)

## [回文数](https://leetcode-cn.com/problems/palindrome-number)

## [3的幂](https://leetcode-cn.com/problems/power-of-three)

## [比特位计数](https://leetcode-cn.com/problems/counting-bits)

## [各位相加](https://leetcode-cn.com/problems/add-digits)

## [丑数](https://leetcode-cn.com/problems/ugly-number)

## [丑数 II](https://leetcode-cn.com/problems/ugly-number-ii)

## [缺失数字](https://leetcode-cn.com/problems/missing-number)

## [两整数之和](https://leetcode-cn.com/problems/sum-of-two-integers)

## [查找和最小的K对数字](https://leetcode-cn.com/problems/find-k-pairs-with-smallest-sums)

## [猜数字大小](https://leetcode-cn.com/problems/guess-number-higher-or-lower)

## [猜数字大小 II](https://leetcode-cn.com/problems/guess-number-higher-or-lower-ii)

## [寻找两个有序数组的中位数](https://leetcode-cn.com/problems/median-of-two-sorted-arrays)

## [汉明距离](https://leetcode-cn.com/problems/hamming-distance)

## [第三大的数](https://leetcode-cn.com/problems/third-maximum-number)

利用了set的唯一性和有序性

class Solution {

public:

    int thirdMax(vector<int>& nums) {

        set<int> s;

        for (int i : nums) {

            s.insert(i);

            if (s.size() > 3) s.erase(s.begin());//删除小的，类似最小堆，但是是唯一的，因为排序了

        }

        return s.size() == 3 ? \*s.begin() : \*s.rbegin();

    }

};

## [数字的补数](https://leetcode-cn.com/problems/number-complement)

## [七进制数](https://leetcode-cn.com/problems/base-7)

## [学生出勤记录 I](https://leetcode-cn.com/problems/student-attendance-record-i)

## [自除数](https://leetcode-cn.com/problems/self-dividing-numbers)

## [宝石与石头](https://leetcode-cn.com/problems/jewels-and-stones)

## [机器人能否返回原点](https://leetcode-cn.com/problems/robot-return-to-origin)

# 数组

(排列组合)

## [第k个排列](https://leetcode-cn.com/problems/permutation-sequence)

## [目标和](https://leetcode-cn.com/problems/target-sum)

## [四数相加 II](https://leetcode-cn.com/problems/4sum-ii)

## [下一个排列](https://leetcode-cn.com/problems/next-permutation)

（区间和）

[和为K的子数组](https://leetcode-cn.com/problems/subarray-sum-equals-k)

(其他数组)

## [在排序数组中查找元素的第一个和最后一个位置](https://leetcode-cn.com/problems/find-first-and-last-position-of-element-in-sorted-array)

## [寻找重复数](https://leetcode-cn.com/problems/find-the-duplicate-number)

## [除自身以外数组的乘积](https://leetcode-cn.com/problems/product-of-array-except-self)

## [区域和检索 - 数组不可变](https://leetcode-cn.com/problems/range-sum-query-immutable)

## [汇总区间](https://leetcode-cn.com/problems/summary-ranges)

## [最长公共前缀](https://leetcode-cn.com/problems/longest-common-prefix)

跟第一个比较

class Solution {

public:

    string longestCommonPrefix(vector<string>& strs) {

        if(strs.size() == 0)return "";

        int n = strs.size();

        const string& firstStr = strs[0];

        int len = firstStr.size();//最长前缀的长度

        for(int i = 1; i < n; i++)

        {

            int k = 0;

            for(; k < len && k < strs[i].size(); k++)

                if(firstStr[k] != strs[i][k])break;

            if(k < len)len = k;

        }

        return firstStr.substr(0, len);

    }

};

## [逆波兰表达式求值](https://leetcode-cn.com/problems/evaluate-reverse-polish-notation)

class Solution {

public:

int evalRPN(vector<string>& tokens) {

stack<long> sk;//长整形栈

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

{

if (tokens[i].size() == 1 && tokens[i][0] == '+')//符号则弹出前面两个并处理，逆波兰的特性

{

long x1 = sk.top();

sk.pop();

long x2 = sk.top();

sk.pop();

sk.push(x2 + x1);

}

else if (tokens[i].size() == 1 && tokens[i][0] == '-')

{

long x1 = sk.top();

sk.pop();

long x2 = sk.top();

sk.pop();

sk.push(x2 - x1);

}

else if (tokens[i].size() == 1 && tokens[i][0] == '\*')

{

long x1 = sk.top();

sk.pop();

long x2 = sk.top();

sk.pop();

sk.push(x2 \* x1);

}

else if (tokens[i].size() == 1 && tokens[i][0] == '/')

{

long x1 = sk.top();

sk.pop();

long x2 = sk.top();

sk.pop();

sk.push(x2 / x1);

}

else//数字就压入等待处理

{

if (tokens[i][0] == '-')

{

long x = strtol(tokens[i].c\_str()+1,NULL,10);

sk.push(-x);

}

else

{

long x = strtol(tokens[i].c\_str(),NULL,10);

sk.push(x);

}

}

}

return sk.top();

}

};

## [寻找峰值](https://leetcode-cn.com/problems/find-peak-element)

## [同构字符串](https://leetcode-cn.com/problems/isomorphic-strings)

## [颜色分类](https://leetcode-cn.com/problems/sort-colors)

class Solution {

public:

void sortColors(vector<int>& nums) {

// 荷兰国旗问题，需要三个指针

int begin = 0, current = 0, end = nums.size() - 1;

while (current <= end){

if (nums[current] == 0){

swap(nums[current++], nums[begin++]);

}

else if (nums[current] == 2){

swap(nums[current], nums[end--]);

}

else {

current++;

}

}

}

};

## [两数之和 II - 输入有序数组](https://leetcode-cn.com/problems/two-sum-ii-input-array-is-sorted)

## [加油站](https://leetcode-cn.com/problems/gas-station)

class Solution {

public:

    int canCompleteCircuit(vector<int>& gas, vector<int>& cost) {

        int n = gas.size();

        if (n == 0)return -1;

        for(int i = 0;i < n;)

        {

            int g(0);

            int j = i;

            for(;j < i + n;++j)//循环遍历

            {

                g += gas[j%n] - cost[j%n];

                if (g < 0)break;

            }

            if (g >= 0)return i;

            if (j < n && j > i + 1) i = j;//跳过已经验证过的

            else ++i;

        }

        return -1;

    }

};

## [旋转数组](https://leetcode-cn.com/problems/rotate-array)

翻转会调换位置，再翻转回来就行

class Solution {

public:

    void rotate(vector<int>& nums, int k) {

        if (nums.size() <= 1)return;

        k = k % nums.size();

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

        reverse(nums.begin(),nums.begin()+k);

        reverse(nums.begin()+k,nums.end());

    }

};

## [删除排序数组中的重复项 II](https://leetcode-cn.com/problems/remove-duplicates-from-sorted-array-ii)

## [求众数](https://leetcode-cn.com/problems/majority-element)

## [杨辉三角](https://leetcode-cn.com/problems/pascals-triangle)

## [杨辉三角 II](https://leetcode-cn.com/problems/pascals-triangle-ii)

## [合并两个有序数组](https://leetcode-cn.com/problems/merge-sorted-array)

## [搜索二维矩阵](https://leetcode-cn.com/problems/search-a-2d-matrix)

把二维矩阵当做一维数组来计算，就是访问下标时需要转换下标

class Solution {

public:

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

        if (!matrix.size() || !matrix[0].size())return false;

        int rows=matrix.size();

        int cols=matrix[0].size();

        int left=0,right=(rows\*cols-1);

        int mid,m,n,val;

        while(left <= right){

            mid= (right+left)>>1;

            m= mid / cols;

            n= mid % cols;

            if(matrix[m][n]==target) return true;

            else if (matrix[m][n] < target) left = mid+1;

            else right= mid - 1;

        }

        return false;

    }

};

## [螺旋矩阵](https://leetcode-cn.com/problems/spiral-matrix)

注意边界条件

class Solution {

public:

    vector<int> spiralOrder(vector<vector<int>>& matrix) {

       if (matrix.size() == 0)return {};

        vector<int> ret;

        int row = matrix.size() - 1;

        int col = matrix[0].size() - 1;

        for (int x = 0, y = 0; x <= row && y <= col; x++, y++,row--,col--)

        {

            for(int j=y ; j<=col ; ++j)//首行

            {

                ret.push\_back(matrix[x][j]);

            }

            for (int i = x + 1; i <= row; ++i)//最右列(上到下，需要跳过第一个)

            {

                ret.push\_back(matrix[i][col]);

            }

            for (int j = col - 1; j >= y && x != row; --j)//最底行(右到左，需要跳过第一个，判断重复行)

            {

                ret.push\_back(matrix[row][j]);

            }

            for (int i = row - 1; i > x && y != col; --i)//最左列(下到上，需要跳过第一个，判断重复列)

            {

                ret.push\_back(matrix[i][y]);

            }

        }

        return ret;

    }

};

## [螺旋矩阵 II](https://leetcode-cn.com/problems/spiral-matrix-ii)

## [搜索旋转排序数组](https://leetcode-cn.com/problems/search-in-rotated-sorted-array)

## [搜索旋转排序数组 II](https://leetcode-cn.com/problems/search-in-rotated-sorted-array-ii)

## [数组中的第K个最大元素](https://leetcode-cn.com/problems/kth-largest-element-in-an-array)

class Solution {

public:

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

        priority\_queue<int,vector<int>,greater<int>> pq;//最小堆

        for(auto & num:nums)

        {

            pq.push(num);

            if (pq.size() > k) pq.pop();

        }

        return pq.top();

    }

};

## [存在重复元素](https://leetcode-cn.com/problems/contains-duplicate)

哈希表

class Solution {

public:

    bool containsDuplicate(vector<int>& nums) {

        unordered\_map<int,int> m;

        for(auto& num:nums)

        {

            if (m[num]++ > 0)return true;

        }

        return false;

    }

};

## [存在重复元素 II](https://leetcode-cn.com/problems/contains-duplicate-ii)

遍历时检查下标差

class Solution {

public:

    bool containsNearbyDuplicate(vector<int>& nums, int k) {

        unordered\_map<int,int> intmap;

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

        {

            if(intmap.end() != intmap.find(nums[i]) &&(i - intmap[nums[i]]) <= k)

            {

                return true;

            }

            else

            {

                intmap[nums[i]] = i;

            }

        }

        return false;

    }

};

## [存在重复元素 III](https://leetcode-cn.com/problems/contains-duplicate-iii)

利用set的有序性计算键的差值，消除非法下标，就可以获取合法条件值

class Solution {

public:

    bool containsNearbyAlmostDuplicate(vector<int>& nums, int k, int t) {

        set<long> st;

        int j = 0;

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

            if (i - j > k) st.erase(nums[j++]);//消除下标非法位置的

            auto iter = st.lower\_bound((long)nums[i] - t);//获取和小于等于t的下一个数（n >= nums[i] - t)  ,nums[i] - n <= t） ，再去判断其绝对值

            if (iter != st.end() && abs(\*iter - nums[i]) <= t) return true;

            st.insert(nums[i]);

        }

        return false;

    }

};

## [长度最小的子数组](https://leetcode-cn.com/problems/minimum-size-subarray-sum)

## [合并区间](https://leetcode-cn.com/problems/merge-intervals)

先排序，再遍历检查和合并

class Solution {

public:

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

        if (intervals.size()<=1)  return  intervals;

        vector<Interval> res;

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

        Interval pre = intervals[0] ;

        for (int index=1; index<intervals.size(); index++)

        {

            Interval tmp = intervals[index] ;

            if (tmp.start > pre.end)

            {

                res.push\_back(pre);

                pre = tmp;

            }

            else

            {

                pre.end = max(tmp.end, pre.end);

            }

        }

        res.push\_back(pre);

        return res;

    }

    static bool cmp(const Interval &val1, const Interval &val2){

        return val1.start < val2.start;

    }

};

## [插入区间](https://leetcode-cn.com/problems/insert-interval)

## [删除排序数组中的重复项](https://leetcode-cn.com/problems/remove-duplicates-from-sorted-array)

## [递增的三元子序列](https://leetcode-cn.com/problems/increasing-triplet-subsequence)

## [滑动窗口最大值](https://leetcode-cn.com/problems/sliding-window-maximum)

使用链表保存窗口中的从大到小的列表，表头的就是需要的值

class Solution {

public:

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

        if (nums.size() == 0 || !k) return {};

        vector<int> res;

        list<int> l;//链表表头始终保存的是最大的元素，递减较小的成员会排在后面作为替补成员

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

        {

            if (l.size() && l.front() == i - k)l.pop\_front();

            while(l.size() && nums[l.back()] < nums[i])

            {

                l.pop\_back();

            }

            l.push\_back(i);

            if (i >= k - 1)res.push\_back(nums[l.front()]);

        }

        return res;

    }

};

## [移动零](https://leetcode-cn.com/problems/move-zeroes)

从前往后边遍历交换，保持稳定性

class Solution {

public:

    void moveZeroes(vector<int>& nums) {

        int k = 0;

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

        {

            if (nums[i])

            {

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

            }

        }

    }

};

## [计算右侧小于当前元素的个数](https://leetcode-cn.com/problems/count-of-smaller-numbers-after-self)

## [打乱数组](https://leetcode-cn.com/problems/shuffle-an-array)

## [有序矩阵中第K小的元素](https://leetcode-cn.com/problems/kth-smallest-element-in-a-sorted-matrix)

## [摆动排序 II](https://leetcode-cn.com/problems/wiggle-sort-ii)

## [组合总和 Ⅳ](https://leetcode-cn.com/problems/combination-sum-iv)

## [找到所有数组中消失的数字](https://leetcode-cn.com/problems/find-all-numbers-disappeared-in-an-array)

## [两个数组的交集](https://leetcode-cn.com/problems/intersection-of-two-arrays)

使用哈希表的快速访问

class Solution {

public:

    /\*

    复杂度为m + n

    \*/

    vector<int> intersection(vector<int>& nums1, vector<int>& nums2) {

        vector<int> res;

        unordered\_set<int> s1(nums1.begin(),nums1.end());

        unordered\_set<int> s2(nums2.begin(),nums2.end());

        for(auto iter:s2)

        {

            if (s1.find(iter) != s1.end())res.push\_back(iter);

        }

        return res;

    }

};

## [两个数组的交集 II](https://leetcode-cn.com/problems/intersection-of-two-arrays-ii)

## [最大连续1的个数](https://leetcode-cn.com/problems/max-consecutive-ones)

遍历数组和判断

class Solution {

public:

    int findMaxConsecutiveOnes(vector<int>& nums) {

        int maxLen(0);

        int left(-1);

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

        {

            if (1 == nums[i])maxLen = max(maxLen,i-left);

            else left = i;

        }

        return maxLen;

    }

};

## [连续的子数组和](https://leetcode-cn.com/problems/continuous-subarray-sum)

## [数组拆分 I](https://leetcode-cn.com/problems/array-partition-i)

## [错误的集合](https://leetcode-cn.com/problems/set-mismatch)

## [数组的度](https://leetcode-cn.com/problems/degree-of-an-array)

## [划分为k个相等的子集](https://leetcode-cn.com/problems/partition-to-k-equal-sum-subsets)

## [转置矩阵](https://leetcode-cn.com/problems/transpose-matrix)

## [最长连续递增序列](https://leetcode-cn.com/problems/longest-continuous-increasing-subsequence)

## [最短无序连续子数组](https://leetcode-cn.com/problems/shortest-unsorted-continuous-subarray)

## [任务调度器](https://leetcode-cn.com/problems/task-scheduler)

因为任务的等待时间，制约的就是数量最大的任务，时间=（该类任务等待时间 + 1） \* （该类任务数量 -1） + 该类任务同样数量的任务的类数  ，或者任务数中的较大的

class Solution {

public:

    int leastInterval(vector<char>& tasks, int n) {

        vector<int> mVec(26,0);

        for(auto &val:tasks) {

            mVec[val-'A']++;

        }

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

        int i(0);

        while(i < mVec.size() && mVec.back()==mVec[mVec.size() - 1 - i])i++;

        return max((mVec.back()-1)\*(n+1)+i,(int)tasks.size());

    }

};

# 字符串

(排列)

## [下一个更大元素 III](https://leetcode-cn.com/problems/next-greater-element-iii)

(切分)

## [简化路径](https://leetcode-cn.com/problems/simplify-path)

字节流的分割和遍历

class Solution {

public:

    string simplifyPath(string path) {

        vector<string> vec;

        stringstream ss(path);

        string tmp;

        while(getline(ss,tmp,'/'))//从前往后 取出所有的 子路径

        {

            if(tmp == "" || tmp == ".") continue;

            if(tmp == "..")

            {

                if (vec.size() > 0) vec.pop\_back();

            }

            else vec.push\_back(tmp);

        }

        string res;

        for(const auto &str:vec) res += "/" + str;

        return res.size() ? res:"/";

    }

};

（字符数组）

## [压缩字符串](https://leetcode-cn.com/problems/string-compression)

(其他)

## [基本计算器 II](https://leetcode-cn.com/problems/basic-calculator-ii)

## [最后一个单词的长度](https://leetcode-cn.com/problems/length-of-last-word)

## [实现strStr()](https://leetcode-cn.com/problems/implement-strstr)

## [验证回文串](https://leetcode-cn.com/problems/valid-palindrome)

## [报数](https://leetcode-cn.com/problems/count-and-say)

## [字符串相乘](https://leetcode-cn.com/problems/multiply-strings)

## [找不同](https://leetcode-cn.com/problems/find-the-difference)

## [去除重复字母](https://leetcode-cn.com/problems/remove-duplicate-letters)

贪心计算字符

class Solution {

public:

    string removeDuplicateLetters(string s) {

        int m[256] = {0}, visited[256] = {0};

        string res = "0";

        for (auto a : s) ++m[a];//计数，为了判断字符数

        for (auto a : s) {

            --m[a];

            if (visited[a]) continue;//同样的字符前面访问过的就不再访问，因为没意义

            while (a < res.back() && m[res.back()]) {//较小的，后面还有该字符，则被移除（贪心）

                visited[res.back()] = 0;

                res.pop\_back();

            }

            res += a;//加入新的字符

            visited[a] = 1;

        }

        return res.substr(1);

    }

};

## [最长回文子串](https://leetcode-cn.com/problems/longest-palindromic-substring)

## [反转字符串](https://leetcode-cn.com/problems/reverse-string)

## [反转字符串中的元音字母](https://leetcode-cn.com/problems/reverse-vowels-of-a-string)

## [至少有K个重复字符的最长子串](https://leetcode-cn.com/problems/longest-substring-with-at-least-k-repeating-characters)

## [字符串中的第一个唯一字符](https://leetcode-cn.com/problems/first-unique-character-in-a-string)

## [字符串解码](https://leetcode-cn.com/problems/decode-string)

## [找到字符串中所有字母异位词](https://leetcode-cn.com/problems/find-all-anagrams-in-a-string)

## [无重复字符的最长子串](https://leetcode-cn.com/problems/longest-substring-without-repeating-characters)

## [有效的字母异位词](https://leetcode-cn.com/problems/valid-anagram)

## [Fizz Buzz](https://leetcode-cn.com/problems/fizz-buzz)

## [字符串相加](https://leetcode-cn.com/problems/add-strings)

## [字符串中的单词数](https://leetcode-cn.com/problems/number-of-segments-in-a-string)

## [最长回文串](https://leetcode-cn.com/problems/longest-palindrome)

## [重复的子字符串](https://leetcode-cn.com/problems/repeated-substring-pattern)

## [键盘行](https://leetcode-cn.com/problems/keyboard-row)

## [检测大写字母](https://leetcode-cn.com/problems/detect-capital)

## [单词替换](https://leetcode-cn.com/problems/replace-words)

## [词典中最长的单词](https://leetcode-cn.com/problems/longest-word-in-dictionary)

## [仅仅反转字母](https://leetcode-cn.com/problems/reverse-only-letters)

## [反转字符串 II](https://leetcode-cn.com/problems/reverse-string-ii)

## [反转字符串中的单词 III](https://leetcode-cn.com/problems/reverse-words-in-a-string-iii)

## [实现一个魔法字典](https://leetcode-cn.com/problems/implement-magic-dictionary)

## [验证回文字符串 Ⅱ](https://leetcode-cn.com/problems/valid-palindrome-ii)

## [字母大小写全排列](https://leetcode-cn.com/problems/letter-case-permutation)

## [旋转字符串](https://leetcode-cn.com/problems/rotate-string)

## [字符的最短距离](https://leetcode-cn.com/problems/shortest-distance-to-a-character)

## [验证外星语词典](https://leetcode-cn.com/problems/verifying-an-alien-dictionary)

## [键值映射](https://leetcode-cn.com/problems/map-sum-pairs)

# 二叉树

常用dfs或者bfs，前序、中序、后序遍历，dfs一般需要递归

## [求根到叶子节点数字之和](https://leetcode-cn.com/problems/sum-root-to-leaf-numbers)

## [完全二叉树的节点个数](https://leetcode-cn.com/problems/count-complete-tree-nodes)

## [二叉树中的最大路径和](https://leetcode-cn.com/problems/binary-tree-maximum-path-sum)

## [二叉树的右视图](https://leetcode-cn.com/problems/binary-tree-right-side-view)    bfs

## [路径总和](https://leetcode-cn.com/problems/path-sum)

## [路径总和 II](https://leetcode-cn.com/problems/path-sum-ii)  dfs

//遍历节点，选择或者不选择

class Solution {

public:

    vector<vector<int>> pathSum(TreeNode\* root, int sum) {

        vector<int> path;

        dfs(root,path,0,sum);

        return res;

    }

    void dfs(TreeNode\* root,vector<int> &path,int s ,int sum)

    {

        if (!root)return;

        path.push\_back(root->val);//选择

        if (!root->left && !root->right && root->val + s == sum)

        {

            res.push\_back(path);

            path.pop\_back();

            return;

        }

        dfs(root->left,path,root->val + s,sum);

        dfs(root->right,path,root->val + s,sum);

        path.pop\_back();//不选择，不选择需要放在最后边，因为前面需要遍历

    }

    vector<vector<int>> res;

};

## [二叉树展开为链表](https://leetcode-cn.com/problems/flatten-binary-tree-to-linked-list)

二叉树与链表关系的处理

void flatten(TreeNode\* root) {

        while (root) {

            if (root->left) {

                TreeNode\* pre = root->left;

                while (pre->right)

                    pre = pre->right;

                pre->right = root->right;

                root->right = root->left;

                root->left = NULL;

            }

            root = root->right;

        }

    }

## [填充每个节点的下一个右侧节点指针](https://leetcode-cn.com/problems/populating-next-right-pointers-in-each-node)

## [填充每个节点的下一个右侧节点指针 II](https://leetcode-cn.com/problems/populating-next-right-pointers-in-each-node-ii)

## [二叉树的前序遍历](https://leetcode-cn.com/problems/binary-tree-preorder-traversal)

/\*\*

\* 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:

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

if(!root)return {};

vector<int> res;

stack<TreeNode\*> s;

TreeNode\* p = root;

while(p ||s.size())

{

while(p)

{

res.push\_back(p->val);

s.push(p);

p = p->left;

}

p = s.top();

s.pop();

p = p->right;

}

return res;

}

};

## [二叉树的后序遍历](https://leetcode-cn.com/problems/binary-tree-postorder-traversal)

/\*\*

\* 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:

vector<int> postorderTraversal(TreeNode\* root)

{

if (!root)return {};

TreeNode\* p = root;

vector<int> res;

stack<TreeNode\*> s;

while(p || s.size())

{

while(p)

{

res.push\_back(p->val);//先序

s.push(p);

p = p->right;//跟先序的相反

}

p = s.top();

s.pop();

p = p->left;//跟先序的相反

}

reverse(res.begin(),res.end());//反序

return res;

}

};

## [翻转二叉树](https://leetcode-cn.com/problems/invert-binary-tree)

## [二叉树的序列化与反序列化](https://leetcode-cn.com/problems/serialize-and-deserialize-binary-tree)

//先序遍历序列化，对应的就可以使用dfs来反序列化了

class Codec {

public:

    // Encodes a tree to a single string.

    string serialize(TreeNode\* root)//"[1,2,3,#,#,4,5]"

    {

        if (!root)return "";

        string res("[");

        //先序遍历

        stack<TreeNode\*> sk;

        sk.push(root);

        while(sk.size())

        {

            TreeNode\* p = sk.top();

            sk.pop();

            if (!p)

            {

                res.append("#,");

            }

            else

            {

                res.append(to\_string(p->val) + ",");

                sk.push(p->right);//有否子节点也压入

                sk.push(p->left);

            }

            //printf("%u\n",sk.size());

        }

        res.pop\_back();

        res.append("]");

        //printf("%s\n",res.c\_str());

        return res;

    }

    // Decodes your encoded data to tree.

    TreeNode\* deserialize(string data) {

        if (data.size() <= 2)return NULL;

        vector<string> total;

        data.erase(data.begin());

        data.pop\_back();

        string str;

        stringstream in(data);

        while(getline(in, str,','))

        {

            total.push\_back(str);

        }

        int index(0);

        TreeNode\* root = GetTree(total,index);

        return root;

    }

    TreeNode\* GetTree(vector<string> &total,int &index)//先序遍历的方式解析

    {

        if (index == total.size())return NULL;

        if (total[index] == "#"){++index;return NULL;}//遇到#表示该子树已结束

        //处理当前节点

        int num(0);

        int op = total[index][0] == '-' ? -1:1;

        for(int i = (op == -1) ? 1:0;i < total[index].size();++i)

        {

            num = num \* 10 + total[index][i] - '0';

        }

        TreeNode\* node = new TreeNode(num \* op);

        ++index;

        node->left = GetTree(total,index);

        node->right = GetTree(total,index);

        return node;

    }

};

## [二叉搜索树中第K小的元素](https://leetcode-cn.com/problems/kth-smallest-element-in-a-bst)

因为是二叉搜索树，使用中序遍历就可以，就是从小到大。使用栈来遍历

class Solution {

public:

    int kthSmallest(TreeNode\* root, int k)

    {

        //bst,中序遍历方式

        if (!root)return 0;

        stack<TreeNode\*> sk;

        TreeNode\* p = root;

        int cnt(0);

        while(p || sk.size())

        {

            while(p)

            {

                sk.push(p);

                p = p->left;

            }

            p = sk.top();

            sk.pop();

            if (++cnt == k)return p->val;

            p = p->right;

        }

        return 0;

    }

};

## [二叉搜索树的最近公共祖先](https://leetcode-cn.com/problems/lowest-common-ancestor-of-a-binary-search-tree)

因为是二叉搜索树，则可以根据值选择哪边，在中间的为本节点，大了或小了就选边，然后遍历下去

class Solution {

public:

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

    {

        while(root)

        {

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

            {

                break;

            }

            if (max(p->val,q->val) < root->val)

            {

                root = root->left;

            }

            else if  (min(p->val,q->val) > root->val)

            {

                root = root->right;

            }

            else break;

        }

        return root;

    }

};

## [二叉树的最近公共祖先](https://leetcode-cn.com/problems/lowest-common-ancestor-of-a-binary-tree)

因为是二叉树，求的是最近的，递归下去获取需求的节点，在两边的就是本节点，在一边的就是该边返回的

class Solution {

public:

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

        if (!root || root == p|| root == q) return root;

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

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

        if (left && right) return root;

        return left ? left:right;

    }

};

## [左叶子之和](https://leetcode-cn.com/problems/sum-of-left-leaves)

## [二叉树的中序遍历](https://leetcode-cn.com/problems/binary-tree-inorder-traversal)

## [不同的二叉搜索树 II](https://leetcode-cn.com/problems/unique-binary-search-trees-ii)

## [不同的二叉搜索树](https://leetcode-cn.com/problems/unique-binary-search-trees)

## [验证二叉搜索树](https://leetcode-cn.com/problems/validate-binary-search-tree)

## [恢复二叉搜索树](https://leetcode-cn.com/problems/recover-binary-search-tree)

## [相同的树](https://leetcode-cn.com/problems/same-tree)

## [对称二叉树](https://leetcode-cn.com/problems/symmetric-tree)

## [二叉树的层次遍历](https://leetcode-cn.com/problems/binary-tree-level-order-traversal)

## [二叉树的锯齿形层次遍历](https://leetcode-cn.com/problems/binary-tree-zigzag-level-order-traversal)

## [二叉树的最大深度](https://leetcode-cn.com/problems/maximum-depth-of-binary-tree)

## [从前序与中序遍历序列构造二叉树](https://leetcode-cn.com/problems/construct-binary-tree-from-preorder-and-inorder-traversal)

## [从中序与后序遍历序列构造二叉树](https://leetcode-cn.com/problems/construct-binary-tree-from-inorder-and-postorder-traversal)

## [二叉树的层次遍历 II](https://leetcode-cn.com/problems/binary-tree-level-order-traversal-ii)

## [将有序数组转换为二叉搜索树](https://leetcode-cn.com/problems/convert-sorted-array-to-binary-search-tree)

## [有序链表转换二叉搜索树](https://leetcode-cn.com/problems/convert-sorted-list-to-binary-search-tree)

## [平衡二叉树](https://leetcode-cn.com/problems/balanced-binary-tree)

## [二叉树的最小深度](https://leetcode-cn.com/problems/minimum-depth-of-binary-tree)

## [二叉树的所有路径](https://leetcode-cn.com/problems/binary-tree-paths)

## [路径总和 III](https://leetcode-cn.com/problems/path-sum-iii)

## [找树左下角的值](https://leetcode-cn.com/problems/find-bottom-left-tree-value)

## [二叉搜索树中的众数](https://leetcode-cn.com/problems/find-mode-in-binary-search-tree)

## [N叉树的最大深度](https://leetcode-cn.com/problems/maximum-depth-of-n-ary-tree)

## [二叉树的直径](https://leetcode-cn.com/problems/diameter-of-binary-tree)

## [最长同值路径](https://leetcode-cn.com/problems/longest-univalue-path)

## [两数之和 IV - 输入 BST](https://leetcode-cn.com/problems/two-sum-iv-input-is-a-bst)

## [二叉树最大宽度](https://leetcode-cn.com/problems/maximum-width-of-binary-tree)

## [修剪二叉搜索树](https://leetcode-cn.com/problems/trim-a-binary-search-tree)

## [二叉树的层平均值](https://leetcode-cn.com/problems/average-of-levels-in-binary-tree)

## [二叉搜索树的最小绝对差](https://leetcode-cn.com/problems/minimum-absolute-difference-in-bst)

## [把二叉搜索树转换为累加树](https://leetcode-cn.com/problems/convert-bst-to-greater-tree)

## [二叉搜索树中的搜索](https://leetcode-cn.com/problems/search-in-a-binary-search-tree)

## [二叉搜索树结点最小距离](https://leetcode-cn.com/problems/minimum-distance-between-bst-nodes)

## [二叉树的坡度](https://leetcode-cn.com/problems/binary-tree-tilt)

## [另一个树的子树](https://leetcode-cn.com/problems/subtree-of-another-tree)

## [根据二叉树创建字符串](https://leetcode-cn.com/problems/construct-string-from-binary-tree)

## [合并二叉树](https://leetcode-cn.com/problems/merge-two-binary-trees)

## 