力扣刷题

有的标了留着练手

https://github.com/lyhue1991/eat_pytorch_in_20_days

https://blog.csdn.net/BlckRiver/article/details/105356991 vector几种初始化方式

英文

位运算 Bitwise

中文名称	英文名称	符号
与运算	AND operation	&
或运算	OR operation	`
非运算（取反）	NOT operation	~
异或运算	XOR operation（exclusive OR）	^
左移运算	Left shift	<<
右移运算	Right shift	>>

分类	中文	英文
数组与序列	最长连续序列	Longest Consecutive Sequence
数组与序列	最长递增子序列	Longest Increasing Subsequence
数组与序列	最长公共子序列	Longest Common Subsequence

数组与序列	最长回文子串	Longest Palindromic Substring
数组与序列	合并两个有序数组	Merge Two Sorted Arrays
数组与序列	移动零	Move Zeroes
数组与序列	删除重复元素	Remove Duplicates
哈希与集合	两数之和	Two Sum
哈希与集合	三数之和	3Sum
哈希与集合	字母异位词分组	Group Anagrams
哈希与集合	有效的字母异位词	Valid Anagram
哈希与集合	和为 K 的子数组	Subarray Sum Equals K
字符串处理	字符串相加	Add Strings
字符串处理	实现 strStr()	Implement strStr()
字符串处理	最长公共前缀	Longest Common Prefix
字符串处理	验证回文串	Valid Palindrome
栈与队列	最小栈	Min Stack
栈与队列	用队列实现栈	Implement Stack using Queues
栈与队列	滑动窗口最大值	Sliding Window Maximum
动态规划	爬楼梯	Climbing Stairs
动态规划	打家劫舍	House Robber
动态规划	零钱兑换	Coin Change
动态规划	编辑距离	Edit Distance
图论	图的遍历	Graph Traversal
图论	最短路径	Shortest Path
图论	拓扑排序	Topological Sort
图论	岛屿数量	Number of Islands
图论	课程表	Course Schedule
图论	并查集	Union-Find
回溯	全排列	Permutations
回溯	子集	Subsets
回溯	组合总和	Combination Sum
回溯	解数独	Sudoku Solver
回溯	N 皇后	N-Queens
回溯	单词搜索	Word Search

https://chatgpt.com/c/682cb698-998c-8012-a509-cce1f62c9507

数据结构

并查集：https://algo.itcharge.cn/07.Tree/05.Union-Find/01.Union-Find/

各个数据结构的时间开销

各个数据结构的操作原理和使用方法

常用排序算法

名称	数据对象	稳定性	时间复杂度		额外空间复杂度	描述
			平均	最坏		
冒泡排序	数组	✓	$O(n^2)$		$O(1)$	（无序区，有序区）。 从无序区透过交换找出最大元素放到有序区前端。
选择排序	数组	✗	$O(n^2)$		$O(1)$	（有序区，无序区）。 在无序区里找一个最小的元素跟在有序区的后面。对数组：比较得多，换得少。
	链表	✓				
插入排序	数组、链表	✓	$O(n^2)$		$O(1)$	（有序区，无序区）。 把无序区的第一个元素插入到有序区的合适的位置。对数组：比较得少，换得多。
堆排序	数组	✗	$O(n\log n)$		$O(1)$	（最大堆，有序区）。 从堆顶把根卸出来放在有序区之前，再恢复堆。
归并排序	数组	✓	$O(n\log^2 n)$		$O(1)$	把数据分为两段，从两段中逐个选最小的元素移入新数据段的末尾。 可从上到下或从下到上进行。
			$O(n\log n)$		$O(n)+O(\log n)$ 如果不是从下到上	
	链表				$O(1)$	
快速排序	数组	✗	$O(n\log n)$	$O(n^2)$	$O(\log n)$	（小数，基准元素，大数）。 在区间中随机挑选一个元素作基准，将小于基准的元素放在基准之前，大于基准的元素放在基准之后，再分别对小数区与大数区进行排序。
希尔排序	数组	✗	$O(n\log^2 n)$	$O(n^2)$	$O(1)$	每一轮按照事先决定的间隔进行插入排序，间隔会依次缩小，最后一次一定要是1。
计数排序	数组、链表	✓	$O(n+m)$		$O(n+m)$	统计小于等于该元素值的元素的个数i，于是该元素就放在目标数组的索引i位（i≥0）。
桶排序	数组、链表	✓	$O(n)$		$O(m)$	将值为i的元素放入i号桶，最后依次把桶里的元素倒出来。
基数排序	数组、链表	✓	$O(k\times n)$	$O(n^2)$		一种多关键字的排序算法，可用桶排序实现。

Python

常用操作

Bisect: https://blog.csdn.net/YMWM_/article/details/122378152

代码块

```
sorted_list = sorted(numbers)          # 正向排序：[1, 2, 5, 9]
reverse_sorted = sorted(numbers, reverse=True)  # 反向排序：[9, 5, 2, 1]
numbers.sort() # 原地排序
```

`float('inf')` 和 `float('-inf')`

取数字里不同位，可以直接转成字符串然后当列表来取

注意点

拷贝对象要小心浅拷贝 实现数组赋值时要下面这样

代码块

```python
import copy
b = copy.deepcopy(a)
```

数组与字符串

map使用https://leetcode.cn/problems/majority-element/solutions/146074/duo-shu-yuan-su-by-leetcode-solution（python的话学习counter）

拷贝vector nums.assign(newArr.begin(), newArr.end());

数组轮转三种方法：https://leetcode.cn/problems/rotate-array/solutions/551039/xuan-zhuan-shu-zu-by-leetcode-solution-nipk reverse()

跳跃游戏：https://leetcode.cn/problems/jump-game/solutions/24322/55-by-ikaruga
https://leetcode.cn/problems/jump-game-ii/solutions/36035/45-by-ikaruga

除2可以写移位

88. 合并两个有序数组

代码块

```python
class Solution:
    def merge(self, nums1: List[int], m: int, nums2: List[int], n: int) -> None:
        """
        Do not return anything, modify nums1 in-place instead.
        """
        nums1[m:] = nums2
        nums1.sort()
```

代码块

```cpp
void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {
    int i = nums1.size() - 1;
    m--;
    n--;
    while (n >= 0) {
        while (m >= 0 && nums1[m] > nums2[n]) {
            swap(nums1[i--], nums1[m--]);
        }
        swap(nums1[i--], nums2[n--]);
    }
```

```
11  }
```

80. 删除有序数组中的重复项 II

代码块

```
1   class Solution {
2   public:
3       int removeDuplicates(vector<int>& nums) {
4           return process(nums,2);
5       }
6       int process(vector<int>& nums,int k){
7           int idx = 0;
8           for(auto x : nums){
9               if(idx < k or nums[idx - k] != x){ // 这句好好看看
10                  nums[idx++] = x;
11              }
12          }
13          return idx;
14      }
15  };;
```

169. 多数元素

哈希表

代码块

```
1   class Solution:
2       def majorityElement(self, nums: List[int]) -> int:
3           counts = collections.Counter(nums)
4           print(counts)
5           return max(counts.keys(), key=counts.get)
```

代码块

```
1   class Solution {
2   public:
3       int majorityElement(vector<int>& nums) {
4           unordered_map<int, int> counts;
5           int majority = 0, cnt = 0;
6           for (int num: nums) {
7               ++counts[num];
8               if (counts[num] > cnt) {
9                   majority = num;
```

```
                cnt = counts[num];
            }
        }
        return majority;
    }
};
```

排序 比哈希表更快

代码块

```
class Solution:
    def majorityElement(self, nums: List[int]) -> int:
        nums.sort()
        return nums[len(nums) // 2]
```

代码块

```
class Solution {
public:
    int majorityElement(vector<int>& nums) {
        sort(nums.begin(), nums.end());
        return nums[nums.size() / 2];
    }
};
```

摩尔投票法：

https://leetcode.cn/problems/majority-element/solutions/2362000/169-duo-shu-yuan-su-mo-er-tou-piao-qing-ledrh/?envType=study-plan-v2&envId=top-interview-150

代码块

```
class Solution {
public:
    int majorityElement(vector<int>& nums) {
        int x = 0, votes = 0;
        for (int num : nums){
            if (votes == 0) x = num;
            votes += num == x ? 1 : -1;
        }
        return x;
    }
};
```

189. 轮转数组

代码块

```
用额外数字 学一下assign
class Solution {
public:
    void rotate(vector<int>& nums, int k) {
        int n = nums.size();
        vector<int> newArr(n);
        for (int i = 0; i < n; ++i) {
            newArr[(i + k) % n] = nums[i];
        }
        nums.assign(newArr.begin(), newArr.end());
    }
};
```

代码块

```
翻转
class Solution {
public:
    void reverse(vector<int>& nums, int start, int end) {
        while (start < end) {
            swap(nums[start], nums[end]);
            start += 1;
            end -= 1;
        }
    }

    void rotate(vector<int>& nums, int k) {
        k %= nums.size();
        reverse(nums, 0, nums.size() - 1);
        reverse(nums, 0, k - 1);
        reverse(nums, k, nums.size() - 1);
    }
};
```

代码块

```
python的话reverse要自己写
class Solution:
    def reverse(self, nums: List[int], start: int, end: int):
        while start < end:
            nums[start], nums[end] = nums[end], nums[start]
            start += 1
```

```python
            end -= 1

    def rotate(self, nums: List[int], k: int):
        n = len(nums)
        k %= n
        self.reverse(nums, 0, n - 1)
        self.reverse(nums, 0, k - 1)
        self.reverse(nums, k, n - 1)
```

121. 买卖股票的最佳时机

好好看好好学我擦

代码块

```python
class Solution:
    def maxProfit(self, prices: List[int]) -> int:
        inf = int(1e9)
        minprice = inf
        maxprofit = 0
        for price in prices:
            maxprofit = max(price - minprice, maxprofit)
            minprice = min(price, minprice)
        return maxprofit
```

122. 买卖股票的最佳时机 II

DP

代码块

```java
public class Solution {

    public int maxProfit(int[] prices) {
        int len = prices.length;
        if (len < 2) {
            return 0;
        }

        // cash：持有现金
        // hold：持有股票
        // 状态数组
        // 状态转移：cash → hold → cash → hold → cash → hold → cash
        int[] cash = new int[len];
        int[] hold = new int[len];

        cash[0] = 0;
```

```
        hold[0] = -prices[0];

        for (int i = 1; i < len; i++) {
            // 这两行调换顺序也是可以的
            cash[i] = Math.max(cash[i - 1], hold[i - 1] + prices[i]);
            hold[i] = Math.max(hold[i - 1], cash[i - 1] - prices[i]);
        }
        return cash[len - 1];
    }
```

代码块

```
优化版
public class Solution {

    public int maxProfit(int[] prices) {
        int len = prices.length;
        if (len < 2) {
            return 0;
        }

        // cash：持有现金
        // hold：持有股票
        // 状态转移：cash → hold → cash → hold → cash → hold → cash

        int cash = 0;
        int hold = -prices[0];

        int preCash = cash;
        int preHold = hold;
        for (int i = 1; i < len; i++) {
            cash = Math.max(preCash, preHold + prices[i]);
            hold = Math.max(preHold, preCash - prices[i]);

            preCash = cash;
            preHold = hold;
        }
        return cash;
    }
}
```

贪心

代码块

```
public class Solution {
```

```
2
3       public int maxProfit(int[] prices) {
4           int len = prices.length;
5           if (len < 2) {
6               return 0;
7           }
8
9           int res = 0;
10          for (int i = 1; i < len; i++) {
11              int diff = prices[i] - prices[i - 1];
12              if (diff > 0) {
13                  res += diff;
14              }
15          }
16          return res;
17      }
18  }
```

45 跳跃游戏2

https://leetcode.cn/problems/jump-game-ii/description/?envType=study-plan-v2&envId=top-interview-150

代码块

```
1   class Solution:
2       def jump(self, nums: List[int]) -> int:
3           steps = [1e9]*len(nums)
4           steps[0] = 0
5           for i in range(len(nums)):
6               for dis in range(1, nums[i]+1): # 这里要+1，因为前闭后开
7                   if i + dis < len(nums):
8                       steps[i + dis] = min(steps[i]+1, steps[i + dis])
9           return steps[len(nums)-1]
```

上面这个速度很慢

https://leetcode.cn/problems/jump-game-ii/solutions/36035/45-by-ikaruga

代码块

```
1   int jump(vector<int>& nums)
2   {
3       int ans = 0;
4       int end = 0;
5       int maxPos = 0;
```

```
6       for (int i = 0; i < nums.size() - 1; i++)
7       {
8           maxPos = max(nums[i] + i, maxPos);
9           if (i == end)
10          {
11              end = maxPos;
12              ans++;
13          }
14      }
15      return ans;
```

274. H 指数

主要做法没看懂，就记了个排序吧

代码块

```
1   class Solution:
2       def hIndex(self, citations: List[int]) -> int:
3           sorted_citation = sorted(citations, reverse = True)
4           h = 0; i = 0; n = len(citations)
5           while i < n and sorted_citation[i] > h:
6               h += 1
7               i += 1
8           return h
```

275. H 指数 II

这个二分要学会写！！！！

代码块

```
1   class Solution:
2       def hIndex(self, citations: List[int]) -> int:
3           right = len(citations) - 1
4           left = 0
5           while left <= right:
6               mid = (right + left) // 2
7               if citations[mid] >= len(citations) - mid:
8                   right = mid - 1
9               else:
10                  left = mid + 1
11          return len(citations) - left
```

380 O1时间插入删除

https://leetcode.cn/problems/insert-delete-getrandom-o1/description/?envType=study-plan-v2&envId=top-interview-150

Python字典就当哈希表！

可以不用链表

代码块

```python
class RandomizedSet:
    def __init__(self):
        self.nums = []
        self.indices = {}

    def insert(self, val: int) -> bool:
        if val in self.indices:
            return False
        self.indices[val] = len(self.nums)
        self.nums.append(val)
        return True

    def remove(self, val: int) -> bool:
        if val not in self.indices:
            return False
        id = self.indices[val]
        self.nums[id] = self.nums[-1]
        self.indices[self.nums[id]] = id
        self.nums.pop()
        del self.indices[val]
        return True

    def getRandom(self) -> int:
        return choice(self.nums)
```

代码块

```cpp
class RandomizedSet {
public:
    RandomizedSet() {
        srand((unsigned)time(NULL));
    }

    bool insert(int val) {
        if (indices.count(val)) {
```

```
            return false;
        }
        int index = nums.size();
        nums.emplace_back(val);
        indices[val] = index;
        return true;
    }

    bool remove(int val) {
        if (!indices.count(val)) {
            return false;
        }
        int index = indices[val];
        int last = nums.back();
        nums[index] = last;
        indices[last] = index;
        nums.pop_back();
        indices.erase(val);
        return true;
    }

    int getRandom() {
        int randomIndex = rand()%nums.size();
        return nums[randomIndex];
    }
private:
    vector<int> nums;
    unordered_map<int, int> indices;
};
```

238 自身以外数组乘积

https://leetcode.cn/problems/product-of-array-except-self/?envType=study-plan-v2&envId=top-interview-150

这个下面处理是n-2！！！

代码块

```
class Solution:
    def productExceptSelf(self, nums: List[int]) -> List[int]:
        forward = [nums[0]]
        backward = [nums[-1]]
        n = len(nums)
        for i in range(1, n):
            forward.append(forward[i-1] * nums[i])
            backward.append(backward[i-1] * nums[n-i-1])
```

```
        answer = [backward[n-2]]
        for i in range(1, n-1):
            answer.append(forward[i-1] * backward[n-i-2])
        answer.append(forward[n-2])
        return answer
```

134. 加油站

代码块

```
class Solution:
    def canCompleteCircuit(self, gas: List[int], cost: List[int]) -> int:
        n = len(gas)
        spare = 0
        min_spare = float('inf')
        min_index = 0

        for i in range(n):
            spare += gas[i] - cost[i]
            if spare < min_spare:
                min_spare = spare
                min_index = i

        return ((min_index + 1) % n) if spare >= 0 else -1
```

允许油量为负，但是总剩余油量应该大于等于0，否则不存在解的。存在解的情况下，利用贪心法的思想，找到最低点，它的下一个点出发的话，可以保证前期得到剩余油量最大，所以可以跑完全程。

135 发糖果

https://leetcode.cn/problems/candy/solutions/533150/fen-fa-tang-guo-by-leetcode-solution-f01p/?envType=study-plan-v2&envId=top-interview-150

从左到右再从右到左

代码块

```
class Solution:
    def candy(self, ratings: List[int]) -> int:
        n = len(ratings)
        left = [0] * n
        for i in range(n):
            if i > 0 and ratings[i] > ratings[i - 1]:
```

```
                left[i] = left[i - 1] + 1
            else:
                left[i] = 1

        right = ret = 0
        for i in range(n - 1, -1, -1):
            if i < n - 1 and ratings[i] > ratings[i + 1]:
                right += 1
            else:
                right = 1
            ret += max(left[i], right)

        return ret
```

这个有点抽象

代码块

```
class Solution:
    def candy(self, ratings: List[int]) -> int:
        n = len(ratings)
        ret = 1
        inc, dec, pre = 1, 0, 1

        for i in range(1, n):
            if ratings[i] >= ratings[i - 1]:
                dec = 0
                pre = (1 if ratings[i] == ratings[i - 1] else pre + 1)
                ret += pre
                inc = pre
            else:
                dec += 1
                if dec == inc:
                    dec += 1
                ret += dec
                pre = 1

        return ret
```

依据前面总结的规律，我们可以提出本题的解法。我们从左到右枚举每一个同学，记前一个同学分得的糖果数量为 pre：

如果当前同学比上一个同学评分高，说明我们就在最近的递增序列中，直接分配给该同学 pre+1 个糖果即可。

否则我们就在一个递减序列中，我们直接分配给当前同学一个糖果，并把该同学所在的递减序列中所有的同学都再多分配一个糖果，以保证糖果数量还是满足条件。

我们无需显式地额外分配糖果，只需要记录当前的递减序列长度，即可知道需要额外分配的糖果数量。

同时注意当当前的递减序列长度和上一个递增序列等长时，需要把最近的递增序列的最后一个同学也并进递减序列中。

这样，我们只要记录当前递减序列的长度 `dec`，最近的递增序列的长度 `inc` 和前一个同学分得的糖果数量 `pre` 即可。

42. 接雨水

这个思路有点骚操作 可以再看看代码自己写能不能写对

```java
class Solution {
    public int trap(int[] height) {
        int n = height.length;
        int res = 0;
        // 左右指针：分别指向左右两边界的列
        int left = 0, right = n - 1;
        // 左指针的左边最大高度、右指针的右边最大高度
        int leftMax = height[left], rightMax = height[right];
        // 最两边的列存不了水
        left++;
        right--;
        // 向中间靠拢
        while(left <= right){
            leftMax = Math.max(leftMax, height[left]);
            rightMax = Math.max(rightMax, height[right]);
            if(leftMax < rightMax){
                // 左指针的leftMax比右指针的rightMax矮
                // 说明：左指针的右边至少有一个板子 > 左指针左边所有板子
                // 根据水桶效应，保证了左指针当前列的水量决定权在左边
                // 那么可以计算左指针当前列的水量：左边最大高度-当前列高度
                res += leftMax - height[left];
                left++;
            }else{
                // 右边同理
                res += rightMax - height[right];
                right--;
            }
        }
        return res;
    }
}
```

一层一层扫描 应该是对的但是会超时

代码块

```python
class Solution:
    def trap(self, height: List[int]) -> int:
        highest = max(height)
        water = 0
        for h in range(1, highest+1):
            left_index = -1
            right_index = len(height)
            wall = 0
            for index in range(len(height)-1, -1, -1):
                if height[index] >= h:
                    right_index = index
                    break
            for index in range(0, len(height)):
                if height[index] >= h:
                    left_index = index
                    break
            if right_index == left_index or right_index == left_index + 1:
                continue
            for index in range(left_index+1, right_index):
                if height[index] >= h:
                    wall += 1
            water += right_index - left_index - wall - 1
        return water
```

12. 整数转罗马数字

用符号表

罗马转整数往后看一个来确定加还是减

整数转罗马就把6个特殊情况也编入符号表

58 最后一个单词长度

https://leetcode.cn/problems/length-of-last-word/description/?envType=study-plan-v2&envId=top-interview-150

反向遍历

代码块

```cpp
class Solution {
public:
    int lengthOfLastWord(string s) {
        int index = s.size() - 1;
```

```
5
6           while (s[index] == ' ') {
7               index--;
8           }
9           int wordLength = 0;
10          while (index >= 0 && s[index] != ' ') {
11              wordLength++;
12              index--;
13          }
14
15          return wordLength;
16      }
17  };
```

14. 最长公共前缀

横扫一遍就够 拿第一个单词当最长前缀 然后不断减少

也可以竖向扫描，同时扫所有字符串第i位

这个可以留着练手

151. 反转字符串中的单词

字符串操作复习一下 string可以直接+char

留着练手

```
1   class Solution {
2   public:
3       string reverseWords(string s) {
4           int left = 0, right = s.size() - 1;
5           // 去掉字符串开头的空白字符
6           while (left <= right && s[left] == ' ') ++left;
7
8           // 去掉字符串末尾的空白字符
9           while (left <= right && s[right] == ' ') --right;
10
11          deque<string> d;
12          string word;
13
14          while (left <= right) {
15              char c = s[left];
16              if (word.size() && c == ' ') {
17                  // 将单词 push 到队列的头部
18                  d.push_front(move(word));
19                  word = "";
```

```
            }
            else if (c != ' ') {
                word += c;
            }
            ++left;
        }
        d.push_front(move(word));

        string ans;
        while (!d.empty()) {
            ans += d.front();
            d.pop_front();
            if (!d.empty()) ans += ' ';
        }
        return ans;
    }
};
```

class Solution:

def reverseWords(self, s: str) -> str:

return " ".join(reversed(s.split()))

6. Z 字形变换

可以构造二维 也可以直接这样

```
class Solution {
public:
    string convert(string s, int numRows) {
        if (numRows < 2)
            return s;
        vector<string> rows(numRows);
        int i = 0, flag = -1;
        for (char c : s) {
            rows[i].push_back(c);
            if (i == 0 || i == numRows -1)
                flag = - flag;
            i += flag;
        }
        string res;
        for (const string &row : rows)
            res += row;
        return res;
    }
};
```

28. 找出字符串中第一个匹配项的下标

KMP算法

https://zhuanlan.zhihu.com/p/83334559 实际上next数组在构建状态转移图啊！

68 文本左右对齐

https://leetcode.cn/problems/text-justification/description/?envType=study-plan-v2&envId=top-interview-150

没啥技术含量，主要是折磨？

239. 滑动窗口最大值

很精髓啊 priority_queue用法学习一下

3个方法都挺有意思的

1071. 字符串的最大公因子

由上述性质我们可以先用辗转相除法求得两个字符串长度的最大公约数，取出该长度的前缀串，判断一下它是否能经过若干次拼接得到 str1 和 str2 即可。

代码块

```
class Solution:
    def gcdOfStrings(self, str1: str, str2: str) -> str:
        candidate_len = math.gcd(len(str1), len(str2))
        candidate = str1[: candidate_len]
        if candidate * (len(str1) // candidate_len) == str1 and candidate * (len(str2) // candidate_len) == str2:
            return candidate
        return ''
```

345. 反转字符串中的元音字母

这个虽然简单，但是不要用额外数组存，直接双指针

需要注意python里字符串不可修改，所以要转成数组来做

代码块

```
class Solution:
    def reverseVowels(self, s: str) -> str:
        def isVowel(ch: str) -> bool:
            return ch in "aeiouAEIOU"

        n = len(s)
```

```
        s = list(s)
        i, j = 0, n - 1
        while i < j:
            while i < n and not isVowel(s[i]):
                i += 1
            while j > 0 and not isVowel(s[j]):
                j -= 1
            if i < j:
                s[i], s[j] = s[j], s[i]
                i += 1
                j -= 1

        return "".join(s)
```

334. 递增的三元子序列

思路1：从中间出发，左边找一个小，右边找一个大。可以通过预先计算来优化

代码块

```
class Solution:
    def increasingTriplet(self, nums: List[int]) -> bool:
        n = len(nums)
        if n < 3:
            return False
        leftMin = [0] * n
        leftMin[0] = nums[0]
        for i in range(1, n):
            leftMin[i] = min(leftMin[i - 1], nums[i])
        rightMax = [0] * n
        rightMax[n - 1] = nums[n - 1]
        for i in range(n - 2, -1, -1):
            rightMax[i] = max(rightMax[i + 1], nums[i])
        for i in range(1, n - 1):
            if leftMin[i - 1] < nums[i] < rightMax[i + 1]:
                return True
        return False
```

思路2：一次遍历，维持尽可能小的first和second

初始时，$first = nums[0]$，$second = +\infty$。对于 $1 \le i < n$，当遍历到下标 i 时，令 $num = nums[i]$，进行如下操作：

1. 如果 $num > second$，则找到了一个递增的三元子序列，返回 true；
2. 否则，如果 $num > first$，则将 $second$ 的值更新为 num；
3. 否则，将 $first$ 的值更新为 num。

如果遍历结束时没有找到递增的三元子序列，返回 false。

代码块

```python
class Solution:
    def increasingTriplet(self, nums: List[int]) -> bool:
        n = len(nums)
        if n < 3:
            return False
        first, second = nums[0], float('inf')
        for i in range(1, n):
            num = nums[i]
            if num > second:
                return True
            if num > first:
                second = num
            else:
                first = num
        return False
```

443. 压缩字符串

这题双指针麻烦得很 容易写错

代码块

```python
class Solution:
    def compress(self, chars: List[str]) -> int:
        if len(chars) <= 1:
            return len(chars)
        left = 0
        right = 0
        while left < len(chars) and right < len(chars):
            cur_char = chars[right]
            right += 1
            count = 1
            while(right < len(chars) and chars[right] == cur_char):
                right += 1
                count += 1
```

```
            if count >= 2:
                chars[left] = cur_char
                for num in str(count):
                    left = left + 1
                    chars[left] = num
                left = left + 1
            else:
                chars[left] = cur_char
                left = left + 1
        return left
```

双指针

283. 移动零

发现自己是弱智

167. 两数之和 II - 输入有序数组

这个双指针看起来不对但实际是对的

11. 盛最多水的容器

也是一个看起来不对实际上对 每次就是把短板往中间移

双指针代表的是 可以作为容器边界的所有位置的范围。在一开始，双指针指向数组的左右边界，表示数组中所有的位置都可以作为容器的边界，因为我们还没有进行过任何尝试。在这之后，我们每次将对应的数字较小的那个指针 往 另一个指针 的方向移动一个位置，就表示我们认为 这个指针不可能再作为容器的边界了。

滑动数组

209. 长度最小的子数组

方法二：前缀和 + 二分查找

方法一的时间复杂度是 $O(n^2)$，因为在确定每个子数组的开始下标后，找到长度最小的子数组需要 $O(n)$ 的时间。如果使用二分查找，则可以将时间优化到 $O(\log n)$。

为了使用二分查找，需要额外创建一个数组 sums 用于存储数组 nums 的前缀和，其中 $\text{sums}[i]$ 表示从 $\text{nums}[0]$ 到 $\text{nums}[i-1]$ 的元素和。得到前缀和之后，对于每个开始下标 i，可通过二分查找得到大于或等于 i 的最小下标 $bound$，使得 $\text{sums}[bound] - \text{sums}[i-1] \geq s$，并更新子数组的最小长度（此时子数组的长度是 $bound - (i-1)$）。

```
class Solution {
public:
    int minSubArrayLen(int s, vector<int>& nums) {
```

```
        int n = nums.size();
        if (n == 0) {
            return 0;
        }
        int ans = INT_MAX;
        vector<int> sums(n + 1, 0);
        // 为了方便计算，令 size = n + 1
        // sums[0] = 0 意味着前 0 个元素的前缀和为 0
        // sums[1] = A[0] 前 1 个元素的前缀和为 A[0]
        // 以此类推
        for (int i = 1; i <= n; i++) {
            sums[i] = sums[i - 1] + nums[i - 1];
        }
        for (int i = 1; i <= n; i++) {
            int target = s + sums[i - 1];
            auto bound = lower_bound(sums.begin(), sums.end(), target);
            if (bound != sums.end()) {
                ans = min(ans, static_cast<int>((bound - sums.begin()) - (i - 1)));
            }
        }
        return ans == INT_MAX ? 0 : ans;
    }
};
```

滑动窗口

先扩大到足够大 然后右边往右移 超了就缩左边 再右边往右移

```
class Solution {
public:
    int minSubArrayLen(int s, vector<int>& nums) {
        int n = nums.size();
        if (n == 0) {
            return 0;
        }
        int ans = INT_MAX;
        int start = 0, end = 0;
        int sum = 0;
        while (end < n) {
            sum += nums[end];
            while (sum >= s) {
                ans = min(ans, end - start + 1);
                sum -= nums[start];
                start++;
            }
```

```
            end++;
        }
        return ans == INT_MAX ? 0 : ans;
    }
};
```

30. 串联所有单词的子串

好难 这个勉强看懂https://leetcode.cn/problems/substring-with-concatenation-of-all-words/solutions/3825/chuan-lian-suo-you-dan-ci-de-zi-chuan-by-powcai

```python
class Solution:
    def findSubstring(self, s: str, words: List[str]) -> List[int]:
        from collections import Counter
        if not s or not words:return []
        one_word = len(words[0])
        all_len = len(words) * one_word
        n = len(s)
        words = Counter(words)
        res = []
        for i in range(0, n - all_len + 1):
            tmp = s[i:i+all_len]
            c_tmp = []
            for j in range(0, all_len, one_word):
                c_tmp.append(tmp[j:j+one_word])
            if Counter(c_tmp) == words:
                res.append(i)
        return res
```

矩阵

54. 螺旋矩阵

这个这么简洁太离谱了啊

```cpp
class Solution {
public:
    vector<int> spiralOrder(vector<vector<int>>& matrix) {
        vector <int> ans;
        if(matrix.empty()) return ans; //若数组为空，直接返回答案
        int u = 0; //赋值上下左右边界
        int d = matrix.size() - 1;
        int l = 0;
```

```
        int r = matrix[0].size() - 1;
        while(true)
        {
            for(int i = l; i <= r; ++i) ans.push_back(matrix[u][i]); //向右移动直到最右
            if(++ u > d) break; //重新设定上边界，若上边界大于下边界，则遍历遍历完成，下同
            for(int i = u; i <= d; ++i) ans.push_back(matrix[i][r]); //向下
            if(-- r < l) break; //重新设定有边界
            for(int i = r; i >= l; --i) ans.push_back(matrix[d][i]); //向左
            if(-- d < u) break; //重新设定下边界
            for(int i = d; i >= u; --i) ans.push_back(matrix[i][l]); //向上
            if(++ l > r) break; //重新设定左边界
        }
        return ans;
    }
};
```

48. 旋转图像

找到逻辑

https://leetcode.cn/problems/rotate-image/solutions/526980/xuan-zhuan-tu-xiang-by-leetcode-solution-vu3m/

题解有点六

289. 生命游戏

生命游戏原地算法：用一个新的状态标记

哈希表

205. 同构字符串

基本操作

但是不要用来计数，直接用这个方法更简单

```
class Solution {
public:
    bool isIsomorphic(string s, string t) {
        unordered_map<char, char> s2t;
        unordered_map<char, char> t2s;
```

```
        int len = s.length();
        for (int i = 0; i < len; ++i) {
            char x = s[i], y = t[i];
            if ((s2t.count(x) && s2t[x] != y) || (t2s.count(y) && t2s[y] !=
x)) {
                return false;
            }
            s2t[x] = y;
            t2s[y] = x;
        }
        return true;
    }
};
```

290. 单词规律

切字符串怎么切

```
class Solution {
public:
    bool wordPattern(string pattern, string str) {
        unordered_map<string, char> str2ch;
        unordered_map<char, string> ch2str;
        int m = str.length();
        int i = 0;
        for (auto ch : pattern) {
            if (i >= m) {
                return false;
            }
            int j = i;
            while (j < m && str[j] != ' ') j++;
            const string &tmp = str.substr(i, j - i);
            if (str2ch.count(tmp) && str2ch[tmp] != ch) {
                return false;
            }
            if (ch2str.count(ch) && ch2str[ch] != tmp) {
                return false;
            }
            str2ch[tmp] = ch;
            ch2str[ch] = tmp;
            i = j + 1;
        }
        return i >= m;
    }
```

```
};
```

49. 字母异位词分组

先给string排序 绝了

```
class Solution {
public:
    vector<vector<string>> groupAnagrams(vector<string>& strs) {
        unordered_map<string, vector<string>> mp;
        for (string& str: strs) {
            string key = str;
            sort(key.begin(), key.end());
            mp[key].emplace_back(str);
        }
        vector<vector<string>> ans;
        for (auto it = mp.begin(); it != mp.end(); ++it) {
            ans.emplace_back(it->second);
        }
        return ans;
    }
};
```

1. 两数之和

太妙了！哈希表就行。但是有坑 如果要输出多个的时候 先插入在匹配就会出现2+3 然后3+2

要先匹配再插入，不然会自己和自己匹配！！！

```
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        unordered_map<int, int> hashtable;
        for (int i = 0; i < nums.size(); ++i) {
            auto it = hashtable.find(target - nums[i]);
            if (it != hashtable.end()) {
                return {it->second, i};
            }
            hashtable[nums[i]] = i;
        }
        return {};
    }
};
```

560. 和为 K 的子数组*

两数之和的升级版 用前缀和+哈希表

128. 最长连续序列*

一个很神奇的题目 很神奇的做法 只从最小号记上升序列

```cpp
class Solution {
public:
    int longestConsecutive(vector<int>& nums) {
        unordered_set<int> num_set;
        for (const int& num : nums) {
            num_set.insert(num);
        }

        int longestStreak = 0;

        for (const int& num : num_set) {
            if (!num_set.count(num - 1)) {
                int currentNum = num;
                int currentStreak = 1;

                while (num_set.count(currentNum + 1)) {
                    currentNum += 1;
                    currentStreak += 1;
                }

                longestStreak = max(longestStreak, currentStreak);
            }
        }

        return longestStreak;
    }
};
```

快慢指针

202. 快乐数

判断重复循环时，可以不用哈希表，用快慢指针

https://leetcode.cn/problems/happy-number/solutions/21454/shi-yong-kuai-man-zhi-zhen-si-xiang-zhao-chu-xun-h

```
class Solution {
public:
    int bitSquareSum(int n) {
        int sum = 0;
        while(n > 0)
        {
            int bit = n % 10;
            sum += bit * bit;
            n = n / 10;
        }
        return sum;
    }

    bool isHappy(int n) {
        int slow = n, fast = n;
        do{
            slow = bitSquareSum(slow);
            fast = bitSquareSum(fast);
            fast = bitSquareSum(fast);
        }while(slow != fast);

        return slow == 1;
    }
};
```

区间

228. 汇总区间

https://leetcode.cn/problems/summary-ranges/description/?envType=study-plan-v2&envId=top-interview-150

这么做神奇地不需要对结尾额外处理

而且nums[i] == nums[i - 1] + 1不会产生整数溢出

```
class Solution {
public:
    vector<string> summaryRanges(vector<int>& nums) {
      vector<string> result;
      string res;
      int i = 0,n = nums.size();
```

```
        while(i < n)
        {
            int low = i;
            ++i;
            while( i < n && nums[i] == nums[i - 1] + 1)
            {
                ++i;
            }
            int high = i -1;
            string temp = to_string(nums[low]);
            if(low < high)
            {
                temp.append("->");
                temp.append(to_string(nums[high]));
            }
            result.push_back(move(temp));
        }
        return result;
    }
};
```

56. 合并区间*

用sort函数排序

https://leetcode.cn/problems/merge-intervals/?envType=study-plan-v2&envId=top-interview-150

```

class Solution {
public:
    vector<vector<int>> merge(vector<vector<int>>& intervals) {
        sort(intervals.begin(), intervals.end());
        vector<vector<int>> res;
        res.push_back(intervals[0]);
        for (auto &kv : intervals) {
            if (kv[0] <= res.back()[1]) {
                res.back()[1] = max(res.back()[1], kv[1]);
            } else if (kv[0] > res.back()[1]) {
                res.push_back(kv);
            }
        }
        return res;
    }
};
```

452. 用最少数量的箭引爆气球

https://leetcode.cn/problems/minimum-number-of-arrows-to-burst-balloons/description/?envType=study-plan-v2&envId=top-interview-150

排序加贪心 需要思考一下贪心是正确的

sort里用lambda表达式重写的comp方法

```
//重写排序方法
bool comp(const node &a, const node &b)
{
    //常引用const T &xxx
    return a.first < b.first || a.first == b.first && a.second <= b.second;
    //<代表升序，>代表降序
}

int main()
{
    node a[] = {{1, 2}, {2, 3}, {2, 1}};
    int length = sizeof(a) / sizeof(node);

    sort(a, a + length, comp);
```

```
class Solution {
public:
    int findMinArrowShots(vector<vector<int>>& points) {
        if (points.empty()) {
            return 0;
        }
        sort(points.begin(), points.end(), [](const vector<int>& u, const vector<int>& v) {
            return u[1] < v[1];
        });
        int pos = points[0][1];
        int ans = 1;
        for (const vector<int>& balloon: points) {
            if (balloon[0] > pos) {
                pos = balloon[1];
                ++ans;
            }
        }
        return ans;
    }
};
```

栈

20. 有效的括号

学习一下智能指针和栈，map的用法

unordered_map https://blog.csdn.net/shouhu010/article/details/129795413

map：http://www.noobyard.com/article/p-vqktwkxq-hp.html

判断字符串长度：https://www.codenong.com/5-different-methods-to-find-the-length-of-a-string-in-cplusplus/

```cpp
class Solution {
public:
    bool isValid(string s) {
        int n = s.size();
        if (n % 2 == 1) {
            return false;
        }

        unordered_map<char, char> pairs = {
            {')', '('},
            {']', '['},
            {'}', '{'}
        };
        stack<char> stk;
        for (char ch: s) {
            if (pairs.count(ch)) {
                if (stk.empty() || stk.top() != pairs[ch]) {
                    return false;
                }
                stk.pop();
            }
            else {
                stk.push(ch);
            }
        }
        return stk.empty();
    }
};
```

71. 简化路径

https://leetcode.cn/problems/simplify-path/description/?envType=study-plan-v2&envId=top-interview-150

```cpp
class Solution {
public:
    string simplifyPath(string path) {
        int beg = 1, end = 1;
        vector<string> names;
        stack<string> ans;
        while(beg < path.size())
        {
            end = path.find('/', beg);
            string s = path.substr(beg, end - beg);
            beg = end + 1;
            names.push_back(s);
        }
        for(string name : names)
        {
            if(!name.empty() && name != ".")
                ans.push_back(name);
            else if(name == "..")
            {
                ans.pop();
            }
        }
        string res;
        if(ans.empty())
            res = "/";
        else
        {
            for(string s : ans)
            {
                res += "/" + s;
            }
        }
        return ans;
    }
};
```

150. 逆波兰表达式求值

https://leetcode.cn/problems/evaluate-reverse-polish-notation/description/?envType=study-plan-v2&envId=top-interview-150

遇到括号啥也不干就行……

atoi(token.c_str())

```
class Solution {
public:
    int evalRPN(vector<string>& tokens) {
        stack<int> stk;
        int n=tokens.size();
        for(int i=0;i<n;i++){
            string& token = tokens[i];
            if(isNumber(token)){
                stk.push(atoi(token.c_str()));
            }
            else{
                int num2=stk.top();
                stk.pop();
                int num1=stk.top();
                stk.pop();
                switch(token[0]){
                    case '+':
                        stk.push(num1+num2);
                        break;
                    case '-':
                        stk.push(num1-num2);
                        break;
                    case '*':
                        stk.push(num1*num2);
                        break;
                    case '/':
                        stk.push(num1/num2);
                        break;
                }

            }
        }
        return stk.top();
    }
    bool isNumber(string& token) {
        return !(token == "+" || token == "-" || token == "*" || token == "/");
    }

};
```

224. 基本计算器*

224是简单版，只有加减，思路在于展开所有括号，搞一个东西记下括号前的符号，如果是-就把括号里符号都反。这个用于记录的东西要用栈，因为括号可能嵌套。每次离开括号就弹出

下面这个应该是完整计算器，还没看

https://leetcode.cn/problems/basic-calculator/?envType=study-plan-v2&envId=top-interview-150

```
class Solution {
public:
    struct Data {
        bool is_number {false};
        long long number {0};
        char op {'\0'};
        Data(bool set_is_number, long long set_number, char set_op) : is_number(set_is_number), number(set_number), op(set_op) {};
    };

    int calculate(string s) {
        // 去除所有空格
        std::string new_s1;
        for (char c : s) {
            if (c != ' ') {
                new_s1 += c;
            }
        }

        // (-n) -> (0-n)
        // (+n) -> (0+n) (本题没有，忽略)
        // 开头的-n -> 0-n
        // 开头的+n -> 0+n (本题没有，忽略)
        std::string new_s2;
        int n = new_s1.length();
        for (int i = 0; i < n; ++i) {
            if (new_s1[i] == '-' && (i == 0 || new_s1[i - 1] == '(')) {
                new_s2 += "0-";
            } else {
                new_s2 += new_s1[i];
            }
        }

        // 中缀表达式 -> 后缀表达式
        // 操作符优先级:: ^ (本题没有，忽略) > *、/ (本题没有，忽略) > +、-
        // 左括号、右括号的优先级单独计算，因为无论定义左括号优先级最高，右括号优先级最低，还是左括号优先级最低，右括号优先级最高，在具体计算的不同逻辑中都无法统一处理
```

```
        // 相同优先级条件下，先出现的优先级更高(即，均是+、-，则先出现的比后出现的优先级级高，即相同优先级的运算符，先出现的先计算)
        // 转换过程:
        // 中缀表达式从前向后遍历过程中，保证op_stack的栈顶是当前操作符优先级最高的
        // 即，如果栈为空，或者当前操作符比栈顶操作符优先级高，则入栈
        // 遇到(，则认为优先级最高，无脑入栈
        // 遇到)，则认为优先级最低，不断弹栈到后缀表达式结果datas中，直到遇到(，操作符)不会入栈
        // 如果栈不为空，且当前操作符比栈顶操作符优先级低或相同(优先级相同时，先出现的优先级更高，需要先进行计算)，则不断弹栈到后缀表达式结果datas中，直到弹到栈为空，或当前操作符优先级比栈顶操作符元素的优先级高，或遇到(，弹栈后，将当前操作符压栈，即，该操作符入栈前，一定要保证所有优先级大于等于该操作符(实际等于时，先出现的优先级也要更高，要先计算)的操作符，都要先输出到后缀表达式结果datas中

        // 存储后缀表达式
        std::vector<Data> datas;
        // 存储操作符op的栈
        std::stack<char> op_stack;

        // 中缀表达式 -> 后缀表达式
        // has_number是为了知道最后是否还有数字元素没有加入到datas中，因为每次遇到操作符才将cur_number写入，但是最后结尾有可能是数字，有可能是操作符)，而且数字可能为0，可能非0，无法判断，所以只能引入额外变量标记
        bool has_number = false;
        long long cur_number = 0;
        for (char c : new_s2) {
            if (c >= '0' && c <= '9') {
                // 数字
                has_number = true;
                cur_number = cur_number * 10 + c - '0';
            } else {
                // 操作符
                if (has_number) {
                    // 将上一个数字输出到后缀表达式结果datas中
                    datas.emplace_back(true, cur_number, '\0');
                    cur_number = 0;
                    has_number = false;
                }

                if (c == '(') {
                    // 遇到(，无脑入栈
                    op_stack.emplace(c);
                } else if (c == ')') {
                    // 遇到)，不断弹栈到后缀表达式结果datas中，直到遇到(，操作符)不会入栈
                    while (!op_stack.empty() && op_stack.top() != '(') {
                        char op = op_stack.top();
```

```
74                          op_stack.pop();
75                          datas.emplace_back(false, 0, op);
76                      }
77
78                      // 将'('弹栈
79                      op_stack.pop();
80                  } else if (c == '+' || c == '-') {
81                      if (op_stack.empty() || op_stack.top() == '(') {
82                          // 如果栈为空，或者当前操作符比栈顶操作符优先级高，则入栈
83                          op_stack.emplace(c);
84                      } else {
85                          // 如果栈不为空，且当前操作符比栈顶操作符优先级低或相同(优先级相同时，先出现的优先级更高，需要先进行计算)，则不断弹栈到后缀表达式结果datas中，直到弹到栈为空，或当前操作符优先级比栈顶操作符元素的优先级高，或遇到(，弹栈后，将当前操作符压栈，即，该操作符入栈前，一定要保证所有优先级大于等于该操作符(实际等于时，先出现的优先级也要更高，要先计算)的操作符，都要先输出到后缀表达式结果datas中
86                          while (!op_stack.empty() && (op_stack.top() == '+' || op_stack.top() == '-')) {
87                              // 这里如果遇到(就不要再弹了，说明这些都是在一组()内处理的部分
88                              char op = op_stack.top();
89                              op_stack.pop();
90                              datas.emplace_back(false, 0, op);
91                          }
92
93                          // 将当前操作符压栈
94                          op_stack.emplace(c);
95                      }
96                  }
97              }
98          }
99          if (has_number) {
100             // 如果原中缀表达式最后一个字符不是)，则最后一个数字还没有输出到后缀表达式结果datas中
101             datas.emplace_back(true, cur_number, '\0');
102         }
103         while (!op_stack.empty()) {
104             // 将栈中剩余操作符依次弹栈到后缀表达式结果datas中
105             char op = op_stack.top();
106             op_stack.pop();
107             datas.emplace_back(false, 0, op);
108         }
109
110         // 计算后缀表达式
111         // 此时后缀表达式结果datas中，只包括数字、+、-，不会再存在括号
112
113         // 存储操作数num的栈
```

```
114             std::stack<long long> num_stack;
115 
116             for (const Data& data : datas) {
117                 if (data.is_number) {
118                     // 如果是数字，就压栈
119                     num_stack.emplace(data.number);
120                 } else {
121                     // 如果是操作符，就进行相应计算
122                     // 先弹栈的是右操作数，后弹栈的是左操作数
123                     long long a = num_stack.top();
124                     num_stack.pop();
125                     long long b = num_stack.top();
126                     num_stack.pop();
127                     if (data.op == '+') {
128                         num_stack.emplace(b + a);
129                     } else if (data.op == '-') {
130                         num_stack.emplace(b - a);
131                     }
132                 }
133             }
134 
135 
136 
137             return num_stack.top();
138         }
139     };
```

链表

是否有环：快慢指针，一个移2一个移1，有环的话必定相遇

141. 环形链表

快慢指针，快能到底部就说明没环

代码块

```
1   # Definition for singly-linked list.
2   # class ListNode:
3   #     def __init__(self, x):
4   #         self.val = x
5   #         self.next = None
6   
7   class Solution:
8       def hasCycle(self, head: ListNode) -> bool:
```

```
        if not head or not head.next:
            return False

        slow = head
        fast = head.next

        while slow != fast:
            if not fast or not fast.next:
                return False
            slow = slow.next
            fast = fast.next.next

        return True
```

然后哈希表存seen也可以

代码块

```
class Solution:
    def hasCycle(self, head: ListNode) -> bool:
        seen = set()
        while head:
            if head in seen:
                return True
            seen.add(head)
            head = head.next
        return False
```

2 两数相加

https://leetcode.cn/problems/add-two-numbers/description/?envType=study-plan-v2&envId=top-interview-150

代码块

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode() {}
 *     ListNode(int val) { this.val = val; }
 *     ListNode(int val, ListNode next) { this.val = val; this.next = next; }
 * }
 */
class Solution {
```

```
12
13
14      public ListNode addTwoNumbers(ListNode l1, ListNode l2) {
15          // 创建一个虚拟头节点【不要动他】
16          ListNode dummy = new ListNode(0);
17          // 创建一个指针指向头节点【动他】
18          ListNode cur = dummy;
19          // 定义一个变量表示进位
20          int carry = 0;
21
22          // 遍历两个链表，直到都为空
23          while (l1 != null || l2 != null) {
24              // 取出两个链表的当前值，如果为空则为0【① 取出加数 被加数】
25              int x = l1 == null ? 0 : l1.val;
26              int y = l2 == null ? 0 : l2.val;
27              // 计算两个值的和，加上进位【② 先求和：加数+被加数+进位】
28              int sum = x + y + carry;
29
30              // 更新进位，如果和大于等于10，则进位为1，否则为0【③ 再更新进位carry，只有
    两个取值1,0】
31              carry = sum >= 10 ? 1 : 0;
32              // 创建一个新的节点，存储和的个位数【④ 最后节点存入结果（链表）sum的个位】
33              cur.next = new ListNode(sum % 10);
34
35  //              ==========  循环指针更新     =============
36              // 移动指针到下一个节点
37              cur = cur.next;
38              // 如果链表不为空，则移动到下一个节点
39              if (l1 != null) l1 = l1.next;
40              if (l2 != null) l2 = l2.next;
41          }
42          // 如果最后还有进位，则在末尾添加一个节点【把多的那个进位 存上】
43          if (carry > 0) {
44              cur.next = new ListNode(carry);
45          }
46          // 返回虚拟头节点的下一个节点，即真正的头节点
47          return dummy.next;
48      }
49  }
```

21 合并两个有序链表

错误示范：head = list1会把前一个节点指向这里的给搞没了

代码块

```
1   # Definition for singly-linked list.
```

```
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def mergeTwoLists(self, list1: Optional[ListNode], list2:
Optional[ListNode]) -> Optional[ListNode]:
        head = ListNode()
        real_head = head
        while list1 and list2:
            if list1.val < list2.val:
                head.val = list1.val
                list1 = list1.next
            else:
                head.val = list2.val
                list2 = list2.next
            head.next = ListNode()
            head = head.next
        if list1:
            head = list1
        else:
            head = list2
        return real_head
```

应该这样：

代码块

```
class Solution:
    def mergeTwoLists(self, l1: ListNode, l2: ListNode) -> ListNode:
        prehead = ListNode(-1)

        prev = prehead
        while l1 and l2:
            if l1.val <= l2.val:
                prev.next = l1
                l1 = l1.next
            else:
                prev.next = l2
                l2 = l2.next
            prev = prev.next

        # 合并后 l1 和 l2 最多只有一个还未被合并完，我们直接将链表末尾指向未合并完的链表
即可
        prev.next = l1 if l1 is not None else l2

```

```
        return prehead.next
```

138. 随机链表的复制*

https://leetcode.cn/problems/copy-list-with-random-pointer/description/?envType=study-plan-v2&envId=top-interview-150

0 不用回溯

代码块

```
class Solution {
    public Node copyRandomList(Node head) {
        if (head == null)
            return null;
        Map<Node, Node> map = new HashMap<>(); // 原节点 -> 新节点映射
        Node curr = head;
        // 第一次遍历：创建新节点并建立映射
        while (curr != null) {
            map.put(curr, new Node(curr.val));
            curr = curr.next;
        }
        // 第二次遍历：设置next和random指针
        curr = head;
        while (curr != null) {
            Node clone = map.get(curr);
            clone.next = map.get(curr.next);
            clone.random = map.get(curr.random);
            curr = curr.next;
        }
        return map.get(head);
    }
}
```

1. 回溯加哈希字典

```
class Solution {
public:
    unordered_map<Node*, Node*> cachedNode;

    Node* copyRandomList(Node* head) {
        if (head == nullptr) {
            return nullptr;
        }
        if (!cachedNode.count(head)) {
```

```
            Node* headNew = new Node(head->val);
            cachedNode[head] = headNew;
            headNew->next = copyRandomList(head->next);
            headNew->random = copyRandomList(head->random);
        }
        return cachedNode[head];
    }
};
```

2. 可以一个拆两个，复制再拼回

206 链表反转

https://leetcode.cn/problems/reverse-linked-list/description/

最后要return的pre!!!!!!!!!!!!!!!!!!!!!!!!!!!!

代码块

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def reverseList(self, head: Optional[ListNode]) -> Optional[ListNode]:
        pre = None
        cur = head
        while cur:
            save_next = cur.next
            cur.next = pre
            pre = cur
            cur = save_next
        return pre
```

92. 反转链表 II*

https://leetcode.cn/problems/reverse-linked-list-ii/solutions/634701/fan-zhuan-lian-biao-ii-by-leetcode-solut-teyq/?envType=study-plan-v2&envId=top-interview-150

反转一个区间，就是取出一段来反转

```
class Solution {
private:
    void reverseLinkedList(ListNode *head) {
```

```cpp
        // 也可以使用递归反转一个链表
        ListNode *pre = nullptr;
        ListNode *cur = head;

        while (cur != nullptr) {
            ListNode *next = cur->next;
            cur->next = pre;
            pre = cur;
            cur = next;
        }
    }

public:
    ListNode *reverseBetween(ListNode *head, int left, int right) {
        // 因为头节点有可能发生变化，使用虚拟头节点可以避免复杂的分类讨论
        ListNode *dummyNode = new ListNode(-1);
        dummyNode->next = head;

        ListNode *pre = dummyNode;
        // 第 1 步：从虚拟头节点走 left - 1 步，来到 left 节点的前一个节点
        // 建议写在 for 循环里，语义清晰
        for (int i = 0; i < left - 1; i++) {
            pre = pre->next;
        }

        // 第 2 步：从 pre 再走 right - left + 1 步，来到 right 节点
        ListNode *rightNode = pre;
        for (int i = 0; i < right - left + 1; i++) {
            rightNode = rightNode->next;
        }

        // 第 3 步：切断出一个子链表（截取链表）
        ListNode *leftNode = pre->next;
        ListNode *curr = rightNode->next;

        // 注意：切断链接
        pre->next = nullptr;
        rightNode->next = nullptr;

        // 第 4 步：同第 206 题，反转链表的子区间
        reverseLinkedList(leftNode);

        // 第 5 步：接回到原来的链表中
        pre->next = rightNode;
        leftNode->next = curr;
        return dummyNode->next;
    }
```

```
51  };
```

25 K 个一组翻转链表#

https://leetcode.cn/problems/reverse-nodes-in-k-group/solutions/248591/k-ge-yi-zu-fan-zhuan-lian-biao-by-leetcode-solutio/?envType=study-plan-v2&envId=top-interview-150

这个留着练手。很折磨

代码块

```
1
```

19. 删除链表的倒数第 N 个结点*

https://leetcode.cn/problems/remove-nth-node-from-end-of-list/description/?envType=study-plan-v2&envId=top-interview-150

倒数第N个，所以搞一个快指针快N。否则从第一个开始还要先获取链表长度

就是双指正

82. 删除排序链表中的重复元素 II

一个错误的结果

代码块

```python
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def deleteDuplicates(self, head: Optional[ListNode]) -> Optional[ListNode]:
        if not head:
            return None
        prev = ListNode
        new_head = prev
        prev.next = head
        curr = head
        while curr and curr.next:
            if curr.next.val == curr.val:
                while curr.next.val == curr.val:
                    curr = curr.next
                prev.next = curr.next # 但是这个也可能是重复的
                prev = curr.next
                curr = curr.next.next
```

```
            else:
                prev = curr
                curr = curr.next
        return new_head.next
```

正确示范

核心是设置了标志确定有没有重复 然后curr每次循环都要跳 设置prev.next然后prev没重复才跳到自己next

代码块

```python
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next

class Solution:
    def deleteDuplicates(self, head: Optional[ListNode]) -> Optional[ListNode]:
        dummy = ListNode(0)  # 虚拟头节点
        dummy.next = head
        prev = dummy  # 指向最后一个不重复的节点
        curr = head

        while curr:
            duplicate = False
            # 如果当前节点有重复值，跳过它们
            while curr.next and curr.val == curr.next.val:
                curr = curr.next
                duplicate = True
            if duplicate:
                # 跳过所有重复节点
                prev.next = curr.next
            else:
                # 当前节点无重复，移动 prev
                prev = prev.next
            curr = curr.next


        return dummy.next

```

61. 旋转链表

题目不难 但是有很多要提前返回的条件 少了一个就不对

代码块

```python
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def rotateRight(self, head: Optional[ListNode], k: int) -> Optional[ListNode]:
        if not head:
            return None
        if k == 0:
            return head
        lens = 0
        curr = head
        while curr:
            lens += 1
            tail = curr
            curr = curr.next
        if lens == 1:
            return head
        ak = k % lens
        if ak == 0:
            return head
        curr = head
        for i in range(lens - ak - 1):
            curr = curr.next
        new_head = curr.next
        curr.next = None
        tail.next = head
        return new_head
```

86. 分隔链表*

https://leetcode.cn/problems/partition-list/?envType=study-plan-v2&envId=top-interview-150

直接把前一半里不符合条件的接到末尾去

```cpp
class Solution {
public:
    ListNode* partition(ListNode* head, int x) {
        if(head==nullptr)
        return nullptr;
```

```
        ListNode* dump=new ListNode(-1,head);
        auto pre=dump,tail=head;
        int size=0;
        while(tail->next)
        {
            tail=tail->next;
            size++;
        }
        for(int i=0;i<=size;i++){
            if(pre->next->val>=x)
            {
                tail->next=pre->next;
                pre->next=pre->next->next;
                tail=tail->next;
                tail->next=nullptr;
            }
            else
                pre=pre->next;
        }
        return dump->next;
    }
};
```

上面有点乱

直接搞一个新列表比较好使

代码块

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def partition(self, head: Optional[ListNode], x: int) ->
Optional[ListNode]:
        before_head = ListNode(0)
        after_head = ListNode(0)

        before = before_head
        after = after_head
        current = head

        while current:
            if current.val < x:
                before.next = current
```

```
                before = before.next
            else:
                after.next = current
                after = after.next
            current = current.next

        # 结束时需断开 after 链表尾部，防止形成环
        after.next = None
        before.next = after_head.next

        return before_head.next

```

146. LRU 缓存

一个链表一个哈希表。移到头 = 删掉+添加到头

https://leetcode.cn/problems/lru-cache/description/?envType=study-plan-v2&envId=top-interview-150

```
struct DLinkedNode {
    int key, value;
    DLinkedNode* prev;
    DLinkedNode* next;
    DLinkedNode(): key(0), value(0), prev(nullptr), next(nullptr) {}
    DLinkedNode(int _key, int _value): key(_key), value(_value), prev(nullptr), next(nullptr) {}
};

class LRUCache {
private:
    unordered_map<int, DLinkedNode*> cache;
    DLinkedNode* head;
    DLinkedNode* tail;
    int size;
    int capacity;

public:
    LRUCache(int _capacity): capacity(_capacity), size(0) {
        // 使用伪头部和伪尾部节点
        head = new DLinkedNode();
        tail = new DLinkedNode();
        head->next = tail;
        tail->prev = head;
    }

```

```
26      int get(int key) {
27          if (!cache.count(key)) {
28              return -1;
29          }
30          // 如果 key 存在，先通过哈希表定位，再移到头部
31          DLinkedNode* node = cache[key];
32          moveToHead(node);
33          return node->value;
34      }
35
36      void put(int key, int value) {
37          if (!cache.count(key)) {
38              // 如果 key 不存在，创建一个新的节点
39              DLinkedNode* node = new DLinkedNode(key, value);
40              // 添加进哈希表
41              cache[key] = node;
42              // 添加至双向链表的头部
43              addToHead(node);
44              ++size;
45              if (size > capacity) {
46                  // 如果超出容量，删除双向链表的尾部节点
47                  DLinkedNode* removed = removeTail();
48                  // 删除哈希表中对应的项
49                  cache.erase(removed->key);
50                  // 防止内存泄漏
51                  delete removed;
52                  --size;
53              }
54          }
55          else {
56              // 如果 key 存在，先通过哈希表定位，再修改 value，并移到头部
57              DLinkedNode* node = cache[key];
58              node->value = value;
59              moveToHead(node);
60          }
61      }
62
63      void addToHead(DLinkedNode* node) {
64          node->prev = head;
65          node->next = head->next;
66          head->next->prev = node;
67          head->next = node;
68      }
69
70      void removeNode(DLinkedNode* node) {
71          node->prev->next = node->next;
72          node->next->prev = node->prev;
```

```
    }

    void moveToHead(DLinkedNode* node) {
        removeNode(node);
        addToHead(node);
    }

    DLinkedNode* removeTail() {
        DLinkedNode* node = tail->prev;
        removeNode(node);
        return node;
    }
};
```

Python的话写法要会写，尤其是定义Node

147. 对链表进行插入排序

代码块

```python
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def insertionSortList(self, head: ListNode) -> ListNode:
        if not head:
            return head

        dummyHead = ListNode(0)
        dummyHead.next = head
        lastSorted = head
        curr = head.next

        while curr:
            if lastSorted.val <= curr.val:
                lastSorted = lastSorted.next
            else:
                prev = dummyHead
                while prev.next.val <= curr.val:
                    prev = prev.next
                lastSorted.next = curr.next
                curr.next = prev.next
                prev.next = curr
            curr = lastSorted.next
```

```
27
28          return dummyHead.next
29
```

二叉树

104. 二叉树的最大深度

代码块

```
1   class Solution:
2       def maxDepth(self, root):
3           if root is None:
4               return 0
5           else:
6               left_height = self.maxDepth(root.left)
7               right_height = self.maxDepth(root.right)
8               return max(left_height, right_height) + 1
```

100. 相同的树

代码块

```
1   class Solution:
2       def isSameTree(self, p: TreeNode, q: TreeNode) -> bool:
3           if not p and not q:
4               return True
5           elif not p or not q:
6               return False
7           elif p.val != q.val:
8               return False
9           else:
10              return self.isSameTree(p.left, q.left) and
    self.isSameTree(p.right, q.right)
```

226. 翻转二叉树

https://leetcode.cn/problems/invert-binary-tree/description/?envType=study-plan-v2&envId=top-interview-150

要多搞一个额外变量来存变化后的左右，否则会出现冲突

```
1   class Solution {
2   public:
```

```
    TreeNode* invertTree(TreeNode* root) {
        if(root==nullptr)
            return nullptr;
        auto left=invertTree(root->left);
        auto right=invertTree(root->right);
        root->right=left;
        root->left=right;
        return root;
    }
};
```

101. 对称二叉树

https://leetcode.cn/problems/symmetric-tree/description/?envType=study-plan-v2&envId=top-interview-150

可以递归，也可以用队列，每次放进去两个拿出来两个

代码块

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def check(self, left, right):
        if not left and not right:
            return True
        elif not left or not right:
            return False
        return left.val == right.val and self.check(left.right, right.left) and self.check(left.left, right.right)

    def isSymmetric(self, root: Optional[TreeNode]) -> bool:
        return self.check(root.left, root.right)
```

105. 从前序与中序遍历序列构造二叉树

https://leetcode.cn/problems/construct-binary-tree-from-preorder-and-inorder-traversal/description/?envType=study-plan-v2&envId=top-interview-150

看官方题解

114. 二叉树展开为链表

https://leetcode.cn/problems/flatten-binary-tree-to-linked-list/description/?envType=study-plan-v2&envId=top-interview-150

不用新建，直接递归去变形

```cpp
class Solution {
public:
    void flatten(TreeNode* root) {
        if(root==nullptr)
            return;
        flatten(root->left);
        flatten(root->right);
        auto left=root->left;
        auto right=root->right;
        root->left=nullptr;
        root->right=left;
        auto temp=root;
        while(temp->right)
            temp=temp->right;
        temp->right=right;
    }
};
```

112. 路径总和

https://leetcode.cn/problems/path-sum/description/?envType=study-plan-v2&envId=top-interview-150

路径总和的四种解法：DFS、回溯、BFS、栈

129. 求根节点到叶节点数字之和

https://leetcode.cn/problems/sum-root-to-leaf-numbers/description/?envType=study-plan-v2&envId=top-interview-150

这个简单的递归写法一定要熟练

```cpp
class Solution {
public:
    int dfs(TreeNode* root, int prevSum) {
        if (root == nullptr) {
            return 0;
        }
        int sum = prevSum * 10 + root->val;
        if (root->left == nullptr && root->right == nullptr) {
            return sum;
```

```
        } else {
            return dfs(root->left, sum) + dfs(root->right, sum);
        }
    }
    int sumNumbers(TreeNode* root) {
        return dfs(root, 0);
    }
};
```

124. 二叉树中的最大路径和

https://leetcode.cn/problems/binary-tree-maximum-path-sum/description/?envType=study-plan-v2&envId=top-interview-150

这个递归里的逻辑也是

```
class Solution {
private:
    int maxSum = INT_MIN;

public:
    int maxGain(TreeNode* node) {
        if (node == nullptr) {
            return 0;
        }

        // 递归计算左右子节点的最大贡献值
        // 只有在最大贡献值大于 0 时，才会选取对应子节点
        int leftGain = max(maxGain(node->left), 0);
        int rightGain = max(maxGain(node->right), 0);

        // 节点的最大路径和取决于该节点的值与该节点的左右子节点的最大贡献值
        int priceNewpath = node->val + leftGain + rightGain;

        // 更新答案
        maxSum = max(maxSum, priceNewpath);

        // 返回节点的最大贡献值
        return node->val + max(leftGain, rightGain);
    }

    int maxPathSum(TreeNode* root) {
        maxGain(root);
        return maxSum;
    }
};
```

173. 二叉搜索树迭代器

https://leetcode.cn/problems/binary-search-tree-iterator/description/?envType=study-plan-v2&envId=top-interview-150

复习一下中序遍历怎么写。就是递归时候先左，再自己，再右

```
    void inorder(TreeNode* root, vector<int>& res) {
        if (!root) {
            return;
        }
        inorder(root->left, res);
        res.push_back(root->val);
        inorder(root->right, res);
    }
    vector<int> inorderTraversal(TreeNode* root) {
        vector<int> res;
        inorder(root, res);
        return res;
    }
```

222. 完全二叉树的节点个数

https://leetcode.cn/problems/count-complete-tree-nodes/description/?envType=study-plan-v2&envId=top-interview-150

这个二分的方法去判断有没有那个节点有点精髓 要好好学

```
class Solution {
public:
    int countNodes(TreeNode* root) {
        if (root == nullptr) {
            return 0;
        }
        int level = 0;
        TreeNode* node = root;
        while (node->left != nullptr) {
            level++;
            node = node->left;
        }
        int low = 1 << level, high = (1 << (level + 1)) - 1;
        while (low < high) {
            int mid = (high - low + 1) / 2 + low;
            if (exists(root, level, mid)) {
```

```
                low = mid;
            } else {
                high = mid - 1;
            }
        }
        return low;
    }

    bool exists(TreeNode* root, int level, int k) {
        int bits = 1 << (level - 1);
        TreeNode* node = root;
        while (node != nullptr && bits > 0) {
            if (!(bits & k)) {
                node = node->left;
            } else {
                node = node->right;
            }
            bits >>= 1;
        }
        return node != nullptr;
    }
};
```

236. 二叉树的最近公共祖先

https://leetcode.cn/problems/lowest-common-ancestor-of-a-binary-tree/description/?envType=study-plan-v2&envId=top-interview-150

其实不是很懂这个递归……

```
class Solution {
    public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {
        if (root == null || root == p || root == q) {
            //只要当前根节点是p和q中的任意一个，就返回（因为不能比这个更深了，再深p和q中的一个就没了）
            return root;
        }
        //根节点不是p和q中的任意一个，那么就继续分别往左子树和右子树找p和q
        TreeNode left = lowestCommonAncestor(root.left, p, q);
        TreeNode right = lowestCommonAncestor(root.right, p, q);
        //p和q都没找到，那就没有
        if(left == null && right == null) {
            return null;
```

```
        }
        //左子树没有p也没有q，就返回右子树的结果
        if (left == null) {
            return right;
        }
        //右子树没有p也没有q就返回左子树的结果
        if (right == null) {
            return left;
        }
        //左右子树都找到p和q了，那就说明p和q分别在左右两个子树上，所以此时的最近公共祖先就是root
        return root;
    }
}
```

二叉搜索树

230. 二叉搜索树中第K小的元素

https://leetcode.cn/problems/kth-smallest-element-in-a-bst/description/?envType=study-plan-v2&envId=top-interview-150

感觉这个有点精髓的

98. 验证二叉搜索树

https://leetcode.cn/problems/validate-binary-search-tree/description/?envType=study-plan-v2&envId=top-interview-150

虽然简单，但是递归要写得优雅

```
class Solution {
public:
    bool helper(TreeNode* root, long long lower, long long upper) {
        if (root == nullptr) {
            return true;
        }
        if (root -> val <= lower || root -> val >= upper) {
            return false;
        }
        return helper(root -> left, lower, root -> val) && helper(root -> right, root -> val, upper);
    }
    bool isValidBST(TreeNode* root) {
        return helper(root, LONG_MIN, LONG_MAX);
    }
```

```
};
```

图

所有岛屿问题https://leetcode.cn/problems/number-of-islands/solutions/211211/dao-yu-lei-wen-ti-de-tong-yong-jie-fa-dfs-bian-li-

可以看到，二叉树的 DFS 有两个要素：「**访问相邻结点**」和「**判断 base case**」。

网格结构是「四叉」的。

其次，网格 DFS 中的 base case 是什么？从二叉树的 base case 对应过来，应该是网格中不需要继续遍历、grid[r][c] 会出现数组下标越界异常的格子，也就是那些超出网格范围的格子。

如何避免这样的重复遍历呢？答案是标记已经遍历过的格子。以岛屿问题为例，我们需要在所有值为 1 的陆地格子上做 DFS 遍历。每走过一个陆地格子，就把格子的值改为 2，这样当我们遇到 2 的时候，就知道这是遍历过的格子了。

200. 岛屿数量

https://leetcode.cn/problems/number-of-islands/description/?envType=study-plan-v2&envId=top-interview-150

```cpp
class Solution {
private:
    void dfs(vector<vector<char>>& grid, int r, int c) {
        int nr = grid.size();
        int nc = grid[0].size();

        grid[r][c] = '0';
        if (r - 1 >= 0 && grid[r-1][c] == '1') dfs(grid, r - 1, c);
        if (r + 1 < nr && grid[r+1][c] == '1') dfs(grid, r + 1, c);
        if (c - 1 >= 0 && grid[r][c-1] == '1') dfs(grid, r, c - 1);
        if (c + 1 < nc && grid[r][c+1] == '1') dfs(grid, r, c + 1);
    }

public:
    int numIslands(vector<vector<char>>& grid) {
        int nr = grid.size();
        if (!nr) return 0;
        int nc = grid[0].size();

        int num_islands = 0;
        for (int r = 0; r < nr; ++r) {
            for (int c = 0; c < nc; ++c) {
                if (grid[r][c] == '1') {
                    ++num_islands;
```

```
                    dfs(grid, r, c);
                }
            }
        }

        return num_islands;
    }
};
```

130. 被围绕的区域

代码块

```python
class Solution:
    def solve(self, board: List[List[str]]) -> None:
        if not board:
            return

        m, n = len(board), len(board[0])

        def dfs(x, y):
            if not 0 <= x < m or not 0 <= y < n or board[x][y] != 'O':
                return

            board[x][y] = "A"
            dfs(x + 1, y)
            dfs(x - 1, y)
            dfs(x, y + 1)
            dfs(x, y - 1)

        for i in range(m):
            dfs(i, 0)
            dfs(i, n - 1)

        for i in range(n - 1):
            dfs(0, i)
            dfs(m - 1, i)

        for i in range(m):
            for j in range(n):
                if board[i][j] == "A":
                    board[i][j] = "O"
                elif board[i][j] == "O":
                    board[i][j] = "X"
```

133. 克隆图

https://leetcode.cn/problems/clone-graph/description/?envType=study-plan-v2&envId=top-interview-150

这个DFS非常精髓，好好学

代码块

```
"""
# Definition for a Node.
class Node:
    def __init__(self, val = 0, neighbors = None):
        self.val = val
        self.neighbors = neighbors if neighbors is not None else []
"""

from typing import Optional
class Solution(object):

    def __init__(self):
        self.visited = {}

    def cloneGraph(self, node):
        """
        :type node: Node
        :rtype: Node
        """
        if not node:
            return node

        # 如果该节点已经被访问过了，则直接从哈希表中取出对应的克隆节点返回
        if node in self.visited:
            return self.visited[node]

        # 克隆节点，注意到为了深拷贝我们不会克隆它的邻居的列表
        clone_node = Node(node.val, [])

        # 哈希表存储
        self.visited[node] = clone_node

        # 遍历该节点的邻居并更新克隆节点的邻居列表
        if node.neighbors:
            clone_node.neighbors = [self.cloneGraph(n) for n in node.neighbors]

        return clone_node
```

```cpp
class Solution {
public:
    unordered_map<Node*, Node*> visited;
    Node* cloneGraph(Node* node) {
        if (node == nullptr) {
            return node;
        }

        // 如果该节点已经被访问过了，则直接从哈希表中取出对应的克隆节点返回
        if (visited.find(node) != visited.end()) {
            return visited[node];
        }

        // 克隆节点，注意到为了深拷贝我们不会克隆它的邻居的列表
        Node* cloneNode = new Node(node->val);
        // 哈希表存储
        visited[node] = cloneNode;

        // 遍历该节点的邻居并更新克隆节点的邻居列表
        for (auto& neighbor: node->neighbors) {
            cloneNode->neighbors.emplace_back(cloneGraph(neighbor));
        }
        return cloneNode;
    }
};
```

399. 除法求值

巨难

https://leetcode.cn/problems/evaluate-division/description/?envType=study-plan-v2&envId=top-interview-150

官方解法可以用并查集 超级难啊

然后另一种做法就是建立图 然后每次查询用搜索路径

这个建立图是动态规划 可以好好学学

207. 课程表

建立有向图，每次删掉In-degree为0的节点

也可以深度优先，每次把没有edge往外指的点入栈

代码块

```python
from collections import deque
```

```python

class Solution:
    def canFinish(self, numCourses: int, prerequisites: List[List[int]]) -> bool:
        indegrees = [0 for _ in range(numCourses)]
        adjacency = [[] for _ in range(numCourses)]
        queue = deque()
        # Get the indegree and adjacency of every course.
        for cur, pre in prerequisites:
            indegrees[cur] += 1
            adjacency[pre].append(cur)
        # Get all the courses with the indegree of 0.
        for i in range(len(indegrees)):
            if not indegrees[i]: queue.append(i)
        # BFS TopSort.
        while queue:
            pre = queue.popleft()
            numCourses -= 1
            for cur in adjacency[pre]:
                indegrees[cur] -= 1
                if not indegrees[cur]: queue.append(cur)
        return not numCourses
```

909. 蛇梯棋

https://leetcode.cn/problems/snakes-and-ladders/description/?envType=study-plan-v2&envId=top-interview-150

一个很奇怪的BFS

好好看看题解

433. 最小基因变化

https://leetcode.cn/problems/minimum-genetic-mutation/description/?envType=study-plan-v2&envId=top-interview-150

建邻接表再BFS。理论上不建邻接表也行

代码块

```python
class Solution:
    def minMutation(self, start: str, end: str, bank: List[str]) -> int:
        if start == end:
            return 0

```

```python
        def diffOne(s: str, t: str) -> bool: # 这个函数好好学
            return sum(x != y for x, y in zip(s, t)) == 1

        m = len(bank)
        adj = [[] for _ in range(m)]
        endIndex = -1
        for i, s in enumerate(bank):
            if s == end:
                endIndex = i
            for j in range(i + 1, m):
                if diffOne(s, bank[j]):
                    adj[i].append(j)
                    adj[j].append(i) # 双向的
        if endIndex == -1:
            return -1

        q = [i for i, s in enumerate(bank) if diffOne(start, s)]
        vis = set(q)
        step = 1
        while q:
            tmp = q
            q = []
            for cur in tmp:
                if cur == endIndex:
                    return step
                for nxt in adj[cur]:
                    if nxt not in vis:
                        vis.add(nxt) # 用set来去重，因为是双向的连接
                        q.append(nxt) # 用来判断是否无路可走的
            step += 1
        return -1
```

127. 单词接龙

https://leetcode.cn/problems/word-ladder/solutions/473600/dan-ci-jie-long-by-leetcode-solution/?envType=study-plan-v2&envId=top-interview-150

同样是奇怪的BFS 可以先建立图 然后双向广度优先

和上面基本一样 不过26个单词变化选项比较多 而且可以搞双向

字典树/前缀树

208. 实现 Trie (前缀树)

https://leetcode.cn/problems/implement-trie-prefix-tree/description/?envType=study-plan-v2&envId=top-interview-150

```cpp
class Trie {
private:
    bool isEnd;
    Trie* next[26];
public:
    Trie() {
        isEnd = false;
        memset(next, 0, sizeof(next));
    }

    void insert(string word) {
        Trie* node = this;
        for (char c : word) {
            if (node->next[c-'a'] == NULL) {
                node->next[c-'a'] = new Trie();
            }
            node = node->next[c-'a'];
        }
        node->isEnd = true;
    }

    bool search(string word) {
        Trie* node = this;
        for (char c : word) {
            node = node->next[c - 'a'];
            if (node == NULL) {
                return false;
            }
        }
        return node->isEnd;
    }

    bool startsWith(string prefix) {
        Trie* node = this;
        for (char c : prefix) {
            node = node->next[c-'a'];
            if (node == NULL) {
                return false;
            }
        }
        return true;
    }
};
```

212. 单词搜索 II

https://leetcode.cn/problems/word-search-ii/description/?envType=study-plan-v2&envId=top-interview-150

困难 把要搜的单词建立前缀树 然后去挨个位置DFS （而不是对棋盘建立前缀） 其中标记防止循环 删除已匹配的单词

大厂经典必考题

https://leetcode.cn/problems/word-search-ii/solutions/1000172/dan-ci-sou-suo-ii-by-leetcode-solution-7494

代码块

```
from collections import defaultdict

class Trie:
    def __init__(self):
        self.children = defaultdict(Trie)
        self.word = ""

    def insert(self, word):
        cur = self
        for c in word:
            cur = cur.children[c]
        cur.is_word = True
        cur.word = word

class Solution:
    def findWords(self, board: List[List[str]], words: List[str]) -> List[str]:
        trie = Trie()
        for word in words:
            trie.insert(word)

        def dfs(now, i1, j1):
            if board[i1][j1] not in now.children:
                return

            ch = board[i1][j1]

            now = now.children[ch]
            if now.word != "": # 到结尾了
                ans.add(now.word)
            # 不退出 继续dfs

            board[i1][j1] = "#" # 防止重复
            for i2, j2 in [(i1 + 1, j1), (i1 - 1, j1), (i1, j1 + 1), (i1, j1 - 1)]:
                if 0 <= i2 < m and 0 <= j2 < n:
```

```
                    dfs(now, i2, j2)
            board[i1][j1] = ch # 退出dfs时恢复

        ans = set()
        m, n = len(board), len(board[0])

        for i in range(m):
            for j in range(n):
                dfs(trie, i, j)

        return list(ans)
```

回溯

17. 电话号码的字母组合

https://leetcode.cn/problems/letter-combinations-of-a-phone-number/description/?envType=study-plan-v2&envId=top-interview-150

官方解析好好学 什么是回溯

```
class Solution {
public:
    vector<string> letterCombinations(string digits) {
        vector<string> combinations;
        if (digits.empty()) {
            return combinations;
        }
        unordered_map<char, string> phoneMap{
            {'2', "abc"},
            {'3', "def"},
            {'4', "ghi"},
            {'5', "jkl"},
            {'6', "mno"},
            {'7', "pqrs"},
            {'8', "tuv"},
            {'9', "wxyz"}
        };
        string combination;
        backtrack(combinations, phoneMap, digits, 0, combination);
        return combinations;
    }

    void backtrack(vector<string>& combinations, const unordered_map<char, string>& phoneMap, const string& digits, int index, string& combination) {
```

```
24            if (index == digits.length()) {
25                combinations.push_back(combination);
26            } else {
27                char digit = digits[index];
28                const string& letters = phoneMap.at(digit);
29                for (const char& letter: letters) {
30                    combination.push_back(letter);
31                    backtrack(combinations, phoneMap, digits, index + 1,
    combination);
32                    combination.pop_back();
33                }
34            }
35        }
36    };
```

77. 组合

https://leetcode.cn/problems/combinations/description/?envType=study-plan-v2&envId=top-interview-150

官方解析有一个很离谱的 反向的二进制枚举

代码块

```
1   class Solution {
2       private List<List<Integer>> ans = new ArrayList<>();
3       public List<List<Integer>> combine(int n, int k) {
4           getCombine(n, k, 1, new ArrayList<>());
5           return ans;
6       }
7
8       public void getCombine(int n, int k, int start, List<Integer> list) {
9           if(k == 0) {
10              ans.add(new ArrayList<>(list));
11              return;
12          }
13          for(int i = start;i <= n - k + 1;i++) {
14              list.add(i);
15              getCombine(n, k - 1, i+1, list);
16              list.remove(list.size() - 1);
17          }
18      }
19  }
```

39. 组合总和

https://leetcode.cn/problems/combination-sum/?envType=study-plan-v2&envId=top-interview-150

非常艰难地用left偷到了一点点边界

```cpp
class Solution {
public:
    vector<vector<int>> res;
    vector<vector<int>> combinationSum(vector<int>& candidates, int target) {
        sort(candidates.begin(), candidates.end());
        vector<int> path;
        backtrack(path, target, candidates, 0);
        return res;
    }
    void backtrack(vector<int>& path, int cur, vector<int>& candidates, int left) {
        if (cur == 0){
            res.push_back(path);
        }
        for (int i =left;i<candidates.size();i++) {
            if (cur - candidates[i] <0) break;
            path.push_back(candidates[i]);
            backtrack(path, cur - candidates[i],candidates, i);
            path.pop_back();
        }
    }
};
```

52. N 皇后 II

https://leetcode.cn/problems/n-queens-ii/description/?envType=study-plan-v2&envId=top-interview-150

按行，每行放一个，每次检查有没有冲突

可以用位运算来优化空间复杂度

```cpp
class Solution {
public:
    int totalNQueens(int n) {
        unordered_set<int> columns, diagonals1, diagonals2;
        return backtrack(n, 0, columns, diagonals1, diagonals2);
    }

    int backtrack(int n, int row, unordered_set<int>& columns, unordered_set<int>& diagonals1, unordered_set<int>& diagonals2) {
```

```
        if (row == n) {
            return 1;
        } else {
            int count = 0;
            for (int i = 0; i < n; i++) {
                if (columns.find(i) != columns.end()) {
                    continue;
                }
                int diagonal1 = row - i;
                if (diagonals1.find(diagonal1) != diagonals1.end()) {
                    continue;
                }
                int diagonal2 = row + i;
                if (diagonals2.find(diagonal2) != diagonals2.end()) {
                    continue;
                }
                columns.insert(i);
                diagonals1.insert(diagonal1);
                diagonals2.insert(diagonal2);
                count += backtrack(n, row + 1, columns, diagonals1, diagonals2);
                columns.erase(i);
                diagonals1.erase(diagonal1);
                diagonals2.erase(diagonal2);
            }
            return count;
        }
    }
};
```

22. 括号生成

https://leetcode.cn/problems/generate-parentheses/description/?envType=study-plan-v2&envId=top-interview-150

```
class Solution:
    @lru_cache(None)
    def generateParenthesis(self, n: int) -> List[str]:
        if n == 0:
            return ['']
        ans = []
        for c in range(n):
            for left in self.generateParenthesis(c):
                for right in self.generateParenthesis(n-1-c):
                    ans.append('({}){}'.format(left, right))
```

```
        return ans

```

```
class Solution {
    shared_ptr<vector<string>> cache[100] = {nullptr};
public:
    shared_ptr<vector<string>> generate(int n) {
        if (cache[n] != nullptr)
            return cache[n]; //已经有了的话就不要重复生成了
        if (n == 0) {
            cache[0] = shared_ptr<vector<string>>(new vector<string>{""});
        } else {
            auto result = shared_ptr<vector<string>>(new vector<string>);
            for (int i = 0; i != n; ++i) {
                auto lefts = generate(i);
                auto rights = generate(n - i - 1);
                for (const string& left : *lefts)
                    for (const string& right : *rights)
                        result -> push_back("(" + left + ")" + right);
            }
            cache[n] = result;
        }
        return cache[n];
    }
    vector<string> generateParenthesis(int n) {
        return *generate(n);
    }
};
```

```
class Solution {
        List<String res = new ArrayList<();
        public List<String generateParenthesis(int n) {
            if(n <= 0){
                return res;
            }
            getParenthesis("",n,n);
            return res;
        }

        private void getParenthesis(String str,int left, int right) {
            if(left == 0 && right == 0 ){
                res.add(str);
                return;
```

```
            }
            if(left == right){
                //剩余左右括号数相等，下一个只能用左括号
                getParenthesis(str+"(",left-1,right);
            }else if(left < right){
                //剩余左括号小于右括号，下一个可以用左括号也可以用右括号
                if(left  0){
                    getParenthesis(str+"(",left-1,right);
                }
                getParenthesis(str+")",left,right-1);
            }
        }
    }
```

自己写的

代码块

```python
class Solution:
    def generateParenthesis(self, n: int) -> List[str]:
        ans = []
        def dfs(string, left_count, right_count):
            if len(string)==2*n:
                ans.append(string)
            else:
                if left_count > 0:
                    dfs(string+"(", left_count-1, right_count)
                if right_count > left_count:
                    dfs(string+")", left_count, right_count-1)
            return
        dfs("", n, n)
        return ans
```

79. 单词搜索

https://leetcode.cn/problems/word-search/description/?envType=study-plan-v2&envId=top-interview-150

可以自己写一下当练手

分治

148. 排序链表

https://leetcode.cn/problems/sort-list/description/?envType=study-plan-v2&envId=top-interview-150

自顶向下递归会产生nlogn的空间复杂度，这是因为递归会消耗栈上空间造成的

两种写法可以好好看看题解https://leetcode.cn/problems/sort-list/solutions/492301/pai-xu-lian-biao-by-leetcode-solution

代码块

```python
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def insertionSortList(self, head: ListNode) -> ListNode:
        if not head:
            return head

        dummyHead = ListNode(0)
        dummyHead.next = head
        lastSorted = head
        curr = head.next

        while curr:
            if lastSorted.val <= curr.val:
                lastSorted = lastSorted.next
            else:
                prev = dummyHead
                while prev.next.val <= curr.val:
                    prev = prev.next
                lastSorted.next = curr.next
                curr.next = prev.next
                prev.next = curr
            curr = lastSorted.next

        return dummyHead.next

```

427. 建立四叉树

https://leetcode.cn/problems/construct-quad-tree/?envType=study-plan-v2&envId=top-interview-150

有点抽象

Kadane

53. 最大子数组和

最大连续子数组 顶级题目解析https://leetcode.cn/problems/maximum-subarray/solutions/9058/dong-tai-gui-hua-fen-zhi-fa-python-dai-ma-java-dai

两种动态规划和分治

官方题解很奇怪没懂

918. 环形子数组的最大和

https://leetcode.cn/problems/maximum-sum-circular-subarray/description/?envType=study-plan-v2&envId=top-interview-150

题解里有骚操作 可以取反 选最小

二分查找

35. 搜索插入位置

https://leetcode.cn/problems/search-insert-position/description/?envType=study-plan-v2&envId=top-interview-150

要输出插入的位置似乎并不简单

```cpp
class Solution {
public:
    int searchInsert(vector<int>& nums, int target) {
        int left = 0;
        int right = nums.size()-1;
        int mid = nums.size();
        int ans;
        while(left <= right){
            mid = (left+right)/2;
            if(nums[mid] == target) return mid;
            if(nums[mid] < target){
                left = mid + 1;
                ans = left;
            }
            else{
                right = mid - 1;
            }
        }
        return max(ans, 0);
    }
```

```
};
```

33. 搜索旋转排序数组

https://leetcode.cn/problems/search-in-rotated-sorted-array/description/?envType=study-plan-v2&envId=top-interview-150

还是可以二分 因为还是会有有序的区间

```cpp
class Solution {
public:
    int search(vector<int>& nums, int target) {
        int n = (int)nums.size();
        if (!n) {
            return -1;
        }
        if (n == 1) {
            return nums[0] == target ? 0 : -1;
        }
        int l = 0, r = n - 1;
        while (l <= r) {
            int mid = (l + r) / 2;
            if (nums[mid] == target) return mid;
            if (nums[0] <= nums[mid]) {
                if (nums[0] <= target && target < nums[mid]) {
                    r = mid - 1;
                } else {
                    l = mid + 1;
                }
            } else {
                if (nums[mid] < target && target <= nums[n - 1]) {
                    l = mid + 1;
                } else {
                    r = mid - 1;
                }
            }
        }
        return -1;
    }
};
```

34. 在排序数组中查找元素的第一个和最后一个位置

https://leetcode.cn/problems/find-first-and-last-position-of-element-in-sorted-array/description/?envType=study-plan-v2&envId=top-interview-150

二分到最左边界

```java

  // 两次二分查找，分开查找第一个和最后一个
   // 时间复杂度 O(log n)，空间复杂度 O(1)
   // [1,2,3,3,3,3,4,5,9]
   public int[] searchRange2(int[] nums, int target) {
     int left = 0;
     int right = nums.length - 1;
     int first = -1;
     int last = -1;
     // 找第一个等于target的位置
     while (left <= right) {
       int middle = (left + right) / 2;
       if (nums[middle] == target) {
         first = middle;
         right = middle - 1; //重点
       } else if (nums[middle] > target) {
         right = middle - 1;
       } else {
         left = middle + 1;
       }
     }

     // 最后一个等于target的位置
     left = 0;
     right = nums.length - 1;
     while (left <= right) {
       int middle = (left + right) / 2;
       if (nums[middle] == target) {
         last = middle;
         left = middle + 1; //重点
       } else if (nums[middle] > target) {
         right = middle - 1;
       } else {
         left = middle + 1;
       }
     }

     return new int[]{first, last};
   }
```

153. 寻找旋转排序数组中的最小值

https://leetcode.cn/problems/find-minimum-in-rotated-sorted-array/description/?envType=study-plan-v2&envId=top-interview-150

一个很特别的二分

```java
public static int findMin(int[] nums) {
        int len = nums.length;
        int low = 0;
        int high = len-1;

//        二分查找
        while(low < high){
//            取中间值
            int mid = (high+low)/2;
//            如果中间值小于最大值，则最大值减小
//            疑问：为什么 high = mid;而不是 high = mid-1;
//            解答：{4,5,1,2,3}，如果high=mid-1，则丢失了最小值1
            if (nums[mid] < nums[high]) {
                high = mid;
            } else {
//                如果中间值大于最大值，则最小值变大
//                疑问：为什么 low = mid+1;而不是 low = mid;
//                解答：{4,5,6,1,2,3}，nums[mid]=6，low=mid+1,刚好nums[low]=1
//                继续疑问：上边的解释太牵强了，难道没有可能low=mid+1,正好错过了最小值
//                继续解答：不会错过!!! 如果nums[mid]是最小值的话，则其一定小于nums[high],走if，就不会走else了
                low = mid+1;
            }
        }
//        疑问：为什么while的条件是low<high,而不是low<=high呢
//        解答：low<high，假如最后循环到{*,10,1,*}的这种情况时，nums[low]=10,nums[high]=1,nums[mid]=10,low=mid+1,
//            直接可以跳出循环了,所以low<high,此时low指向的就是最小值的下标;
//            如果low<=high的话，low=high，还会再不必要的循环一次，此时最后一次循环的时候会发生low==high==mid,
//            则nums[mid]==nums[high]，则会走一次else语句，则low=mid+1,此时low指向的是最小值的下一个下标,
//            则需要return[low-1]
        return nums[low];
    }
```

4. 寻找两个正序数组的中位数

https://leetcode.cn/problems/median-of-two-sorted-arrays/description/?envType=study-plan-v2&envId=top-interview-150

非常难的一个二分 看题解吧

堆

215. 数组中的第K个最大元素

https://leetcode.cn/problems/kth-largest-element-in-an-array/solutions/307351/shu-zu-zhong-de-di-kge-zui-da-yuan-su-by-leetcode-/?envType=study-plan-v2&envId=top-interview-150

1 quick select 2 堆排序

这个快排有点没看懂

373. 查找和最小的 K 对数字

https://leetcode.cn/problems/find-k-pairs-with-smallest-sums/description/?envType=study-plan-v2&envId=top-interview-150

先放进去一批，然后每次拿一个就往堆里加下一个

295. 数据流的中位数

https://leetcode.cn/problems/find-median-from-data-stream/description/?envType=study-plan-v2&envId=top-interview-150

https://leetcode.cn/problems/find-median-from-data-stream/solutions/2361972/295-shu-ju-liu-de-zhong-wei-shu-dui-qing-gmdo 看这个题解

非常神奇 搞两个堆一大一小 往A堆加就先加到B再把B顶拿给A 这样确保两边堆顶是中间的

23. 合并 K 个升序链表

位运算/二进制

67. 二进制求和

https://leetcode.cn/problems/add-binary/description/?envType=study-plan-v2&envId=top-interview-150

字符串操作复习一下：

```
1  class Solution {
2  public:
3      string addBinary(string a, string b) {
4          string ans;
5          reverse(a.begin(), a.end());
6          reverse(b.begin(), b.end());
```

```
7
8           int n = max(a.size(), b.size()), carry = 0;
9           for (size_t i = 0; i < n; ++i) {
10              carry += i < a.size() ? (a.at(i) == '1') : 0;
11              carry += i < b.size() ? (b.at(i) == '1') : 0;
12              ans.push_back((carry % 2) ? '1' : '0');
13              carry /= 2;
14          }
15
16          if (carry) {
17              ans.push_back('1');
18          }
19          reverse(ans.begin(), ans.end());
20
21          return ans;
22      }
23  };
```

也可以位运算 但是这个好像很抽象

```
1   class Solution:
2       def addBinary(self, a, b) -> str:
3           x, y = int(a, 2), int(b, 2)
4           while y:
5               answer = x ^ y
6               carry = (x & y) << 1
7               x, y = answer, carry
8           return bin(x)[2:]
```

python还可以转换为10进制来做

代码块

```
1   class Solution:
2       def addBinary(self, a, b) -> str:
3           return '{0:b}'.format(int(a, 2) + int(b, 2))
```

190. 颠倒二进制位

https://leetcode.cn/problems/reverse-bits/description/?envType=study-plan-v2&envId=top-interview-150

逐位颠倒

```cpp
class Solution {
public:
    uint32_t reverseBits(uint32_t n) {
        uint32_t rev = 0;
        for (int i = 0; i < 32 && n > 0; ++i) {
            rev |= (n & 1) << (31 - i);
            n >>= 1;
        }
        return rev;
    }
};
```

pyhton的位运算也是一样的

代码块

```python
class Solution:
    def reverseBits(self, n: int) -> int:
        rev = 0
        for i in range(32):
            rev |= (n & 1) << (31 - i)
            n >>= 1
        return rev
```

位运算分治

若要翻转一个二进制串，可以将其均分成左右两部分，对每部分递归执行翻转操作，然后将左半部分拼在右半部分的后面，即完成了翻转。

```cpp
class Solution {
private:
    const uint32_t M1 = 0x55555555; // 01010101010101010101010101010101
    const uint32_t M2 = 0x33333333; // 00110011001100110011001100110011
    const uint32_t M4 = 0x0f0f0f0f; // 00001111000011110000111100001111
    const uint32_t M8 = 0x00ff00ff; // 00000000111111110000000011111111

public:
    uint32_t reverseBits(uint32_t n) {
        n = n >> 1 & M1 | (n & M1) << 1; //把 n 的所有奇数位移到偶数位，偶数位移到奇数位，完成 1 位组内反转。
        n = n >> 2 & M2 | (n & M2) << 2;
        n = n >> 4 & M4 | (n & M4) << 4;
        n = n >> 8 & M8 | (n & M8) << 8;
        return n >> 16 | n << 16; //交换高 16 位和低 16 位。
    }
```

```
16  };
```

191. 位1的个数

复习一下

这个1左移i位是这么写的!!!!!!!!!!!!!

```
1   class Solution {
2   public:
3       int hammingWeight(uint32_t n) {
4           int ret = 0;
5           for (int i = 0; i < 32; i++) {
6               if (n & (1 << i)) {
7                   ret++;
8               }
9           }
10          return ret;
11      }
12  };
```

有一个定理：观察这个运算：$n \& (n-1)$，其运算结果恰为把 n 的二进制位中的最低位的 1 变为 0 之后的结果。

代码块

```
1   class Solution:
2       def hammingWeight(self, n: int) -> int:
3           ret = 0
4           while n:
5               n &= n - 1
6               ret += 1
7           return ret
```

136. 只出现一次的数字

哈希表或者异或

因此，数组中的全部元素的异或运算结果即为数组中只出现一次的数字。

```
1   class Solution {
2   public:
3       int singleNumber(vector<int>& nums) {
4           int ans = nums[0];
```

```
        if (nums.size() > 1) {
        for (int i = 1; i < nums.size(); i++) {
            ans = ans ^ nums[i];
        }
        }
        return ans;
    }
};
```

代码块

```python
class Solution:
    def singleNumber(self, nums: List[int]) -> int:
        return reduce(lambda x, y: x ^ y, nums)

class Solution:
    def singleNumber(self, nums: List[int]) -> int:
        result = 0
        for num in nums:
            result ^= num  # 等同于 result = result ^ num
        return result
```

137. 只出现一次的数字 II

看题解吧 出现三次的过滤 就是每一位对3取余

具体地，考虑答案的第 i 个二进制位（i 从 0 开始编号），它可能为 0 或 1。对于数组中非答案的元素，每一个元素都出现了 3 次，对应着第 i 个二进制位的 3 个 0 或 3 个 1，无论是哪一种情况，它们的和都是 3 的倍数（即和为 0 或 3）。因此：

答案的第 i 个二进制位就是数组中所有元素的第 i 个二进制位之和除以 3 的余数。

这样一来，对于数组中的每一个元素 x，我们使用位运算 (x >> i) & 1 得到 x 的第 i 个二进制位，并将它们相加再对 3 取余，得到的结果一定为 0 或 1，即为答案的第 i 个二进制位。

- 在模拟 **32 位有符号整数** 的按位操作时，**第 31 位是符号位**
- 如果它是 1，表示结果应该是一个负数
- Python 不会自动处理补码符号，所以我们手动用 `ans -= (1 << 31)` 来设置负数值

代码块

```python
class Solution:
    def singleNumber(self, nums: List[int]) -> int:
        ans = 0
```

```
        for i in range(32):
            total = sum((num >> i) & 1 for num in nums)
            if total % 3:
                # Python 这里对于最高位需要特殊判断
                if i == 31: # 负数
                    ans -= (1 << i)
                else:
                    ans |= (1 << i)
        return ans
```

哈希表写法

代码块

```
class Solution:
    def singleNumber(self, nums: List[int]) -> int:
        freq = collections.Counter(nums)
        ans = [num for num, occ in freq.items() if occ == 1][0]
        return ans
```

201. 数字范围按位与

因为有0的话与出来都是0，对所有数字执行按位与运算的结果是所有对应二进制字符串的公共前缀再用零补上后面的剩余位。所以只要找到最小和最大的公共前缀1

```
class Solution {
public:
    int rangeBitwiseAnd(int m, int n) {
        int shift = 0;
        // 找到公共前缀
        while (m < n) {
            m >>= 1;
            n >>= 1;
            ++shift;
        }
        return m << shift;
    }
};
```

另一种方法就是n&n-1 把大的数右边1不断消 消到小于小的数就是公共前缀

这个不是n循环到m 因为每次都会搞掉一位数 迭代次数比上面那个更少 因为这个只搞1 上面是0和1都要迭代

```
class Solution {
public:
    int rangeBitwiseAnd(int m, int n) {
        while (m < n) {
            // 抹去最右边的 1
            n = n & (n - 1);
        }
        return n;
    }
};
```

数学

9. 回文数

转成字符串后双指针，很简单

或者：

反转一半的数字即可！

```
class Solution {
public:
    bool isPalindrome(int x) {
        // 特殊情况：
        // 如上所述，当 x < 0 时，x 不是回文数。
        // 同样地，如果数字的最后一位是 0，为了使该数字为回文，
        // 则其第一位数字也应该是 0
        // 只有 0 满足这一属性
        if (x < 0 || (x % 10 == 0 && x != 0)) {
            return false;
        }

        int revertedNumber = 0;
        while (x > revertedNumber) {
            revertedNumber = revertedNumber * 10 + x % 10;
            x /= 10;
        }

        // 当数字长度为奇数时，我们可以通过 revertedNumber/10 去除处于中位的数字。
        // 例如，当输入为 12321 时，在 while 循环的末尾我们可以得到 x = 12,
    revertedNumber = 123,
```

```
            // 由于处于中位的数字不影响回文（它总是与自己相等），所以我们可以简单地将其去除。
            return x == revertedNumber || x == revertedNumber / 10;
        }
    };
```

66. 加一

找出第一个不为 999 的元素，将其加一并将后续所有元素置零即可。如果 digits\textit{digits}digits 中所有的元素均为 999，那么对应着「思路」部分的第三种情况，我们需要返回一个新的数组

代码块

```
class Solution {
    public int[] plusOne(int[] digits) {
        for (int i = digits.length - 1; i >= 0; i--) {
            digits[i]++;
            digits[i] = digits[i] % 10;
            if (digits[i] != 0) return digits;
        }
        digits = new int[digits.length + 1];
        digits[0] = 1;
        return digits;
    }
}
```

172. 阶乘后的零

太离谱了

https://leetcode.cn/problems/factorial-trailing-zeroes/solutions/1360892/jie-cheng-hou-de-ling-by-leetcode-soluti-1egk

只要找5的个数就行

代码块

```
class Solution:
    def trailingZeroes(self, n: int) -> int:
        ans = 0
        while n:
            n //= 5
            ans += n
        return ans
```

69. x 的平方根

二分查找/对数换底/牛顿迭代

代码块

```
class Solution(object):
    def mySqrt(self, x):
        """
        :type x: int
        :rtype: int
        """
        low = 0
        high = x
        while low<=high:
            mid =(low+high)//2
            if mid**2<=x:
                ans = mid
                low= mid+1
            else:
                high = mid-1
        return ans
```

50. Pow(x, n)

利用二进制进行快速幂乘法 可以每次除2来递归

结尾还要注意一下幂为负数的情况

代码块

```
class Solution:
    def myPow(self, x: float, n: int) -> float:
        def quickMul(N):
            if N == 0:
                return 1.0
            y = quickMul(N // 2)
            return y * y if N % 2 == 0 else y * y * x

        return quickMul(n) if n >= 0 else 1.0 / quickMul(-n)
```

也可以下面这样直接迭代

```
class Solution {
public:
    double quickMul(double x, long long N) {
        double ans = 1.0;
        // 贡献的初始值为 x
```

```
            double x_contribute = x;
            // 在对 N 进行二进制拆分的同时计算答案
            while (N > 0) {
                if (N % 2 == 1) {
                    // 如果 N 二进制表示的最低位为 1，那么需要计入贡献
                    ans *= x_contribute;
                }
                // 将贡献不断地平方
                x_contribute *= x_contribute;
                // 舍弃 N 二进制表示的最低位，这样我们每次只要判断最低位即可
                N /= 2;
            }
            return ans;
        }

        double myPow(double x, int n) {
            long long N = n;
            return N >= 0 ? quickMul(x, N) : 1.0 / quickMul(x, -N);
        }
    };
```

49. 直线上最多的点数

看题解吧

枚举斜率 需要处理除法精度问题 可以用转化为乘法， 或者字符串+哈希表， 或者?

代码块

```
class Solution:
    def maxPoints(self, points: List[List[int]]) -> int:
        n, ans = len(points), 1
        for i, x in enumerate(points):
            for j in range(i + 1, n):
                y = points[j]
                # 枚举点对 (i,j) 并统计有多少点在该线上，起始 cnt = 2 代表只有 i 和 j 两个点在此线上
                cnt = 2
                for k in range(j + 1, n):
                    p = points[k]
                    s1 = (y[1] - x[1]) * (p[0] - y[0])
                    s2 = (p[1] - y[1]) * (y[0] - x[0])
                    if s1 == s2: cnt += 1
                ans = max(ans, cnt)
        return ans
```

除法结果转字符串来用哈希表

代码块

```python
class Solution:
    def maxPoints(self, points):
        def gcd(a, b):
            return a if b == 0 else gcd(b, a % b)

        n, ans = len(points), 1
        for i in range(n):
            mapping = {}
            maxv = 0
            for j in range(i + 1, n):
                x1, y1 = points[i]
                x2, y2 = points[j]
                a, b = x1 - x2, y1 - y2
                k = gcd(a, b)
                key = str(a // k) + "_" + str(b // k)
                mapping[key] = mapping.get(key, 0) + 1 #处理key不存在的情况!!!!!
                maxv = max(maxv, mapping[key])
            ans = max(ans, maxv + 1)
        return ans
```

一维动态规划

70. 爬楼梯

斐波那契->矩阵快速幂->解数列通项

198. 打家劫舍

写的非常好的动态规划思路

这个逻辑很简单但是是对的

开头几个手动处理

https://leetcode.cn/problems/house-robber/solutions/138131/dong-tai-gui-hua-jie-ti-si-bu-zou-xiang-jie-cjavap

139. 单词拆分

这个动态规划思路值得学习

300. 最长递增子序列

这个状态转移方程要学习

```cpp
class Solution {
public:
    int lengthOfLIS(vector<int>& nums) {
        int n = (int)nums.size();
        if (n == 0) {
            return 0;
        }
        vector<int> dp(n, 0);
        for (int i = 0; i < n; ++i) {
            dp[i] = 1;
            for (int j = 0; j < i; ++j) {
                if (nums[j] < nums[i]) {
                    dp[i] = max(dp[i], dp[j] + 1);
                }
            }
        }
        return *max_element(dp.begin(), dp.end());
    }
};
```

1143. 最长公共子序列

递归+查表

代码块

```python
class Solution:
    def longestCommonSubsequence(self, text1: str, text2: str) -> int:
        memo = {}
        def LCS(i, j):
            if (i, j) in memo:
                return memo[(i, j)]
            if i == len(text1) or j == len(text2):
                return 0
            if text1[i] == text2[j]:
                memo[(i, j)] = 1 + LCS(i + 1, j + 1)
                return memo[(i, j)]
            else:
                memo[(i, j)] = max(LCS(i + 1, j), LCS(i, j + 1))
                return memo[(i, j)]

        return LCS(0, 0)

GPT说也可以直接用内置的哈希表
class Solution:
    def longestCommonSubsequence(self, text1: str, text2: str) -> int:
        from functools import lru_cache
```

```
22
23          @lru_cache(None)
24          def LCS(i, j):
25              if i == len(text1) or j == len(text2):
26                  return 0
27              if text1[i] == text2[j]:
28                  return 1 + LCS(i + 1, j + 1)
29              else:
30                  return max(LCS(i + 1, j), LCS(i, j + 1))
31
32          return LCS(0, 0)
```

动态规划

- 当 $i = 0$ 时，$text_1[0:i]$ 为空，空字符串和任何字符串的最长公共子序列的长度都是 0，因此对任意 $0 \le j \le n$，有 $dp[0][j] = 0$；
- 当 $j = 0$ 时，$text_2[0:j]$ 为空，同理可得，对任意 $0 \le i \le m$，有 $dp[i][0] = 0$。

因此动态规划的边界情况是：当 $i = 0$ 或 $j = 0$ 时，$dp[i][j] = 0$。

当 $i > 0$ 且 $j > 0$ 时，考虑 $dp[i][j]$ 的计算：

- 当 $text_1[i-1] = text_2[j-1]$ 时，将这两个相同的字符称为公共字符，考虑 $text_1[0:i-1]$ 和 $text_2[0:j-1]$ 的最长公共子序列，再增加一个字符（即公共字符）即可得到 $text_1[0:i]$ 和 $text_2[0:j]$ 的最长公共子序列，因此 $dp[i][j] = dp[i-1][j-1] + 1$。
- 当 $text_1[i-1] \ne text_2[j-1]$ 时，考虑以下两项：
  - $text_1[0:i-1]$ 和 $text_2[0:j]$ 的最长公共子序列；
  - $text_1[0:i]$ 和 $text_2[0:j-1]$ 的最长公共子序列。

  要得到 $text_1[0:i]$ 和 $text_2[0:j]$ 的最长公共子序列，应取两项中的长度较大的一项，因此 $dp[i][j] = \max(dp[i-1][j], dp[i][j-1])$。

代码块

```
1   class Solution:
2       def longestCommonSubsequence(self, text1: str, text2: str) -> int:
3           m, n = len(text1), len(text2)
4           dp = [[0] * (n + 1) for _ in range(m + 1)]
5
```

```
        for i in range(1, m + 1):
            for j in range(1, n + 1):
                if text1[i - 1] == text2[j - 1]:
                    dp[i][j] = dp[i - 1][j - 1] + 1
                else:
                    dp[i][j] = max(dp[i - 1][j], dp[i][j - 1])

        return dp[m][n]
```

多维动态规划

120. 三角形最小路径和

可以把空间优化到2n，再优化到n

代码块

```
class Solution:
    def minimumTotal(self, triangle: List[List[int]]) -> int:
        n = len(triangle)
        res = triangle[0][:]
        for i in range(1, n):
            new_res = [0] * (i + 1) # 确保长度够用
            new_res[0] = res[0] + triangle[i][0]
            new_res[i] = res[-1] + triangle[i][-1] # 注意头和尾都要处理
            for j in range(1, i):
                new_res[j] = min(res[j - 1], res[j]) + triangle[i][j]
            res = new_res # 用new_res 而不是last_res，从而防止同个列表对象被改
        return min(res)
```

64. 最小路径和

这个比较简单 优化的话还可以直接改原数组

代码块

```
class Solution:
    def minPathSum(self, grid: List[List[int]]) -> int:
        m = len(grid)
        n = len(grid[0])
        res = [[999]*n for _ in range(m)]
        res[0][0] = grid[0][0]
        for j in range(1, n):
            res[0][j] = res[0][j-1] + grid[0][j]
        for i in range(1, m):
            for j in range(n):
```

```
                if j == 0:
                    res[i][j] = res[i-1][j] + grid[i][j]
                else:
                    res[i][j] = min(res[i-1][j] + grid[i][j], res[i][j-1] +
    grid[i][j])
        return res[-1][-1]
```

63. 不同路径 II

有障碍的话，直接障碍当0即可。这样不会走过

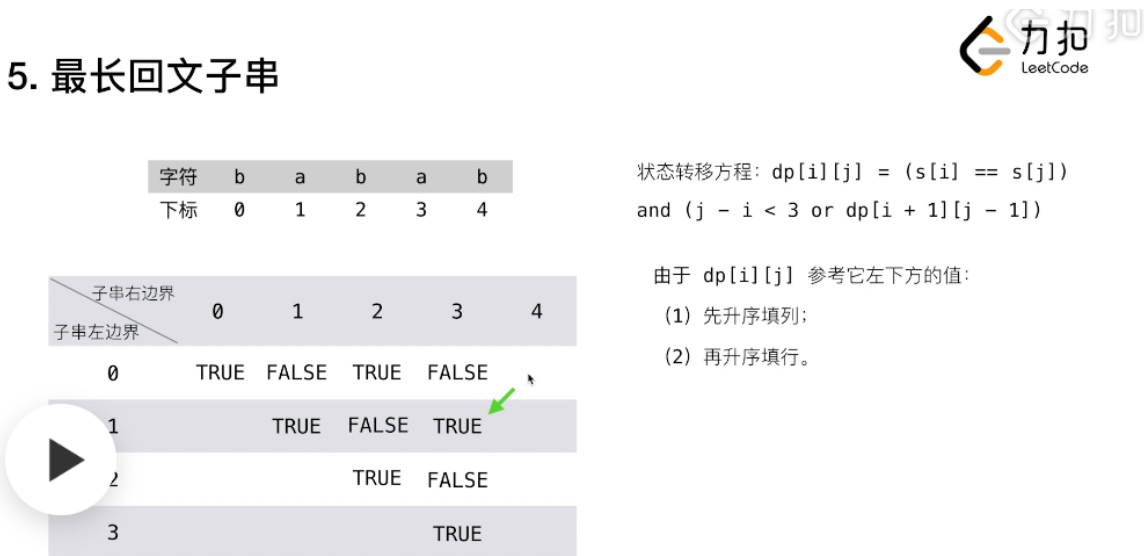
这个空间优化还是要看一下的，从上一行获取已经被隐藏地包含了

代码块

```
class Solution:
    def uniquePathsWithObstacles(self, obstacleGrid: List[List[int]]) -> int:
        n, m = len(obstacleGrid), len(obstacleGrid[0])
        f = [0] * m
        f[0] = 1 if obstacleGrid[0][0] == 0 else 0
        for i in range(n):
            for j in range(m):
                if obstacleGrid[i][j] == 1:
                    f[j] = 0
                elif j > 0 and obstacleGrid[i][j - 1] == 0:
                    f[j] += f[j - 1]
        return f[-1]
```

5. 最长回文子串

这个好好学



那么我们就可以写出动态规划的状态转移方程：

$$P(i,j) = P(i+1,j-1) \wedge (S_i == S_j)$$

也就是说，只有 $s[i+1:j-1]$ 是回文串，并且 s 的第 i 和 j 个字母相同时，$s[i:j]$ 才会是回文串。

上文的所有讨论是建立在子串长度大于 2 的前提之上的，我们还需要考虑动态规划中的边界条件，即子串的长度为 1 或 2。对于长度为 1 的子串，它显然是个回文串；对于长度为 2 的子串，只要它的两个字母相同，它就是一个回文串。因此我们就可以写出动态规划的边界条件：

$$\begin{cases} P(i,i) = \text{true} \\ P(i,i+1) = (S_i == S_{i+1}) \end{cases}$$

代码块

```python
class Solution:
    def longestPalindrome(self, s: str) -> str:
        n = len(s)
        if n < 2:
            return s

        max_len = 1
        begin = 0
        # dp[i][j] 表示 s[i..j] 是否是回文串
        dp = [[False] * n for _ in range(n)]
        for i in range(n):
            dp[i][i] = True

        # 递推开始
        # 先枚举子串长度
        for L in range(2, n + 1):
            # 枚举左边界，左边界的上限设置可以宽松一些
            for i in range(n):
                # 由 L 和 i 可以确定右边界，即 j - i + 1 = L 得
                j = L + i - 1
                # 如果右边界越界，就可以退出当前循环
                if j >= n:
                    break

                if s[i] != s[j]:
                    dp[i][j] = False
                else:
                    if j - i < 3:
                        dp[i][j] = True
                    else:
                        dp[i][j] = dp[i + 1][j - 1]

```

```
                # 只要 dp[i][L] == true 成立，就表示子串 s[i..L] 是回文，此时记录回文长度和起始位置
                if dp[i][j] and j - i + 1 > max_len:
                    max_len = j - i + 1
                    begin = i
        return s[begin:begin + max_len]
```

也可以中间扩散

97. 交错字符串

不能双指针，因为相同字符时无法确定选哪一边的

解决这个问题的正确方法是动态规划。 首先如果 $|s_1| + |s_2| \neq |s_3|$，那 s_3 必然不可能由 s_1 和 s_2 交错组成。在 $|s_1| + |s_2| = |s_3|$ 时，我们可以用动态规划来求解。我们定义 $f(i,j)$ 表示 s_1 的前 i 个元素和 s_2 的前 j 个元素是否能交错组成 s_3 的前 $i+j$ 个元素。如果 s_1 的第 i 个元素和 s_3 的第 $i+j$ 个元素相等，那么 s_1 的前 i 个元素和 s_2 的前 j 个元素是否能交错组成 s_3 的前 $i+j$ 个元素取决于 s_1 的前 $i-1$ 个元素和 s_2 的前 j 个元素是否能交错组成 s_3 的前 $i+j-1$ 个元素，即此时 $f(i,j)$ 取决于 $f(i-1,j)$，在此情况下如果 $f(i-1,j)$ 为真，则 $f(i,j)$ 也为真。同样的，如果 s_2 的第 j 个元素和 s_3 的第 $i+j$ 个元素相等并且 $f(i,j-1)$ 为真，则 $f(i,j)$ 也为真。于是我们可以推导出这样的动态规划转移方程：

$$f(i,j) = [f(i-1,j) \text{ and } s_1(i-1) = s_3(p)] \text{ or } [f(i,j-1) \text{ and } s_2(j-1) = s_3(p)]$$

其中 $p = i + j - 1$。边界条件为 $f(0,0) = \text{True}$。至此，我们很容易可以给出这样一个实现：

代码块

```
class Solution {
public:
    bool isInterleave(string s1, string s2, string s3) {
        auto f = vector < vector <int> > (s1.size() + 1, vector <int> (s2.size() + 1, false));

        int n = s1.size(), m = s2.size(), t = s3.size();

        if (n + m != t) {
            return false;
        }

        f[0][0] = true;
        for (int i = 0; i <= n; ++i) {
            for (int j = 0; j <= m; ++j) {
                int p = i + j - 1;
```

```
                if (i > 0) {
                    f[i][j] |= (f[i - 1][j] && s1[i - 1] == s3[p]);
                }
                if (j > 0) {
                    f[i][j] |= (f[i][j - 1] && s2[j - 1] == s3[p]);
                }
            }
        }

        return f[n][m];
    }
};
```

滚动数组版本

代码块

```
class Solution:
    def isInterleave(self, s1: str, s2: str, s3: str) -> bool:
        n, m, t = len(s1), len(s2), len(s3)
        if n + m != t:
            return False

        f = [False] * (m + 1)
        f[0] = True

        for i in range(n + 1):
            for j in range(m + 1):
                p = i + j - 1
                if i > 0: # 为了滚动数组，这俩要分开写
                    f[j] = f[j] and s1[i - 1] == s3[p]  # 滚动数组后这个要用and f[j]来获取上一行
                if j > 0:
                    f[j] = f[j] or (f[j - 1] and s2[j - 1] == s3[p]) # 滚动数组后这个要用or f[j]来获取上一行

        return f[m]
```

72. 编辑距离

非常逆天

https://leetcode.cn/problems/edit-distance/solutions/6455/zi-di-xiang-shang-he-zi-ding-xiang-xia-by-powcai-3

这样以来，本质不同的操作实际上只有三种：

- 在单词 `A` 中插入一个字符；
- 在单词 `B` 中插入一个字符；
- 修改单词 `A` 的一个字符。

123. 买卖股票的最佳时机 III

https://leetcode.cn/problems/best-time-to-buy-and-sell-stock-iii/solutions/3641314/ling-ji-chu-gao-dong-ti-jie-cong-chuan-t-mw64/?envType=study-plan-v2&envId=top-interview-150 这个题解写的不错

代码块

```cpp
class Solution {
public:
    int maxProfit(vector<int>& prices) {
        int k = 2;
        int n = prices.size();
        vector<int> buy(k, -prices[0]), sell(k, 0);

        for (int i = 1; i < n; i++) {
            for (int j = k - 1; j >= 0; j--) {
                sell[j] = max(buy[j] + prices[i], sell[j]);
                buy[j] = max(j == 0 ? -prices[i] : sell[j - 1] - prices[i] , buy[j]);
            }
        }
        return sell[k - 1];
    }
};
```

188. 买卖股票的最佳时机 IV

给你一个整数数组 `prices` 和一个整数 `k` ，其中 `prices[i]` 是某支给定的股票在第 `i` 天的价格。

设计一个算法来计算你所能获取的最大利润。你最多可以完成 `k` 笔交易。也就是说，你最多可以买 `k` 次，卖 `k` 次。

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）。

多次交易的通解 逆天

关键就是看官方题解，学会表达状态

代码块

```
class Solution:
    def maxProfit(self, k: int, prices: List[int]) -> int:
        if not prices:
            return 0

        n = len(prices)
        k = min(k, n // 2)
        buy = [[0] * (k + 1) for _ in range(n)]
        sell = [[0] * (k + 1) for _ in range(n)]

        buy[0][0], sell[0][0] = -prices[0], 0
        for i in range(1, k + 1):
            buy[0][i] = sell[0][i] = float("-inf")

        for i in range(1, n):
            buy[i][0] = max(buy[i - 1][0], sell[i - 1][0] - prices[i])
            for j in range(1, k + 1):
                buy[i][j] = max(buy[i - 1][j], sell[i - 1][j] - prices[i])
                sell[i][j] = max(sell[i - 1][j], buy[i - 1][j - 1] +
prices[i]);

        return max(sell[n - 1])
```

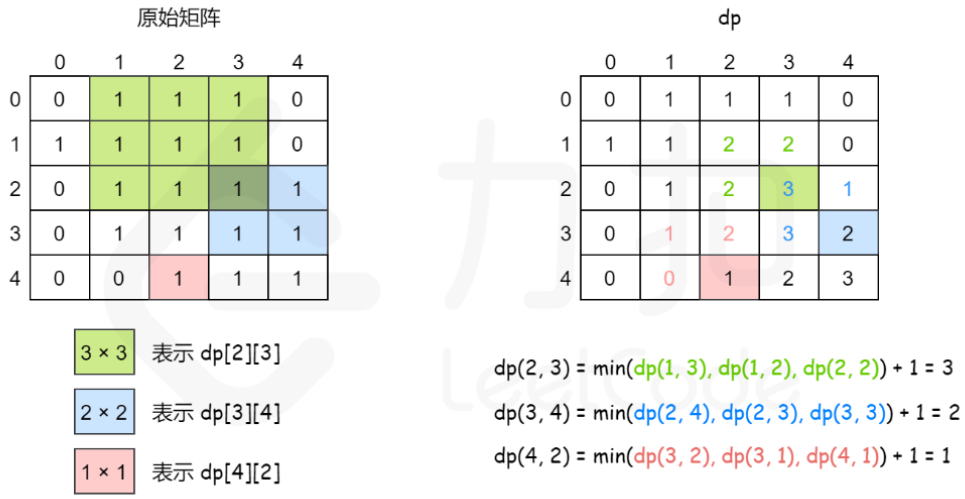
滚动数组优化

代码块

```
class Solution:
    def maxProfit(self, k: int, prices: List[int]) -> int:
        if not prices:
            return 0

        n = len(prices)
        k = min(k, n // 2)
        buy = [0] * (k + 1)
        sell = [0] * (k + 1)

        buy[0], sell[0] = -prices[0], 0
        for i in range(1, k + 1):
            buy[i] = sell[i] = float("-inf")

        for i in range(1, n):
            buy[0] = max(buy[0], sell[0] - prices[i])
            for j in range(1, k + 1):
                buy[j] = max(buy[j], sell[j] - prices[i])
                sell[j] = max(sell[j], buy[j - 1] + prices[i]);
```

```
20
21          return max(sell)
```

221. 最大正方形



暴力法是最简单直观的做法，具体做法如下：

遍历矩阵中的每个元素，每次遇到 1，则将该元素作为正方形的左上角；

确定正方形的左上角后，根据左上角所在的行和列计算可能的最大正方形的边长（正方形的范围不能超出矩阵的行数和列数），在该边长范围内寻找只包含 1 的最大正方形；

每次在下方新增一行以及在右方新增一列，判断新增的行和列是否满足所有元素都是 1。

动态规划：

我们用 *dp*(*i*,*j*) 表示以 (*i*,*j*) 为右下角，且只包含 1 的正方形的边长最大值。

那么如何计算 dp 中的每个元素值呢？对于每个位置 (i,j)，检查在矩阵中该位置的值：

- 如果该位置的值是 0，则 $dp(i,j)=0$，因为当前位置不可能在由 1 组成的正方形中；
- 如果该位置的值是 1，则 $dp(i,j)$ 的值由其上方、左方和左上方的三个相邻位置的 dp 值决定。具体而言，当前位置的元素值等于三个相邻位置的元素中的最小值加 1，状态转移方程如下：

$$dp(i,j)=min(dp(i-1,j),dp(i-1,j-1),dp(i,j-1))+1$$

这个状态转移很妙，只要考虑周围，就相当于考虑整行

代码块

```
1   class Solution:
2       def maximalSquare(self, matrix: List[List[str]]) -> int:
```

```
        if len(matrix) == 0 or len(matrix[0]) == 0:
            return 0

        maxSide = 0
        rows, columns = len(matrix), len(matrix[0])
        dp = [[0] * columns for _ in range(rows)]
        for i in range(rows):
            for j in range(columns):
                if matrix[i][j] == '1':
                    if i == 0 or j == 0:
                        dp[i][j] = 1
                    else:
                        dp[i][j] = min(dp[i - 1][j], dp[i][j - 1], dp[i - 1][j - 1]) + 1
                    maxSide = max(maxSide, dp[i][j])

        maxSquare = maxSide * maxSide
        return maxSquare
```