# 常用代码

* [1. 数据结构](#1-数据结构)
* [1.1. 单例模式](#11-单例模式)
* [1.2. 排序](#12-排序)
  + [1.2.1. 快速排序partition过程](#121-快速排序partition过程)
  + [1.2.2. 快速排序](#122-快速排序)
  + [1.2.3. 非递归](#123-非递归)
  + [1.2.4. 插入排序](#124-插入排序)
  + [1.2.5. 冒泡排序](#125-冒泡排序)
  + [1.2.6. 选择排序](#126-选择排序)
  + [1.2.7. 归并排序](#127-归并排序)
  + [1.2.8. 堆排序](#128-堆排序)
* [1.3. 链表](#13-链表)
  + [1.3.1. 反转链表（头插法）](#131-反转链表头插法)
  + [1.3.2. 单链表去重](#132-单链表去重)
  + [1.3.3. 合并两个链表](#133-合并两个链表)
  + [1.3.4. 链表排序](#134-链表排序)
  + [1.3.5. 检查回文链表](#135-检查回文链表)
  + [1.3.6. 链表等概率随机抽取元素](#136-链表等概率随机抽取元素)
  + [1.3.7. 链表按奇数序号和偶数序号重排](#137-链表按奇数序号和偶数序号重排)
* [1.4. 二叉树遍历](#14-二叉树遍历)
  + [1.4.1. 二叉树前序遍历](#141-二叉树前序遍历)
  + [1.4.2. 二叉树中序遍历](#142-二叉树中序遍历)
  + [1.4.3. 二叉树后序遍历](#143-二叉树后序遍历)
  + [1.4.4. 二叉树层次遍历](#144-二叉树层次遍历)
* [1.5. 图](#15-图)
  + [1.5.1. Dijkstra算法](#151-dijkstra算法)
  + [1.5.2. Floyd-Warshall算法](#152-floyd-warshall算法)
  + [1.5.3. 拓扑排序](#153-拓扑排序)
* [1.6. 查找](#16-查找)
  + [1.6.1. 二分搜索](#161-二分搜索)
  + [1.6.2. 并查集](#162-并查集)
  + [1.6.3. Trie树](#163-trie树)
* [1.7. 回溯](#17-回溯)
  + [1.7.1. N皇后问题](#171-n皇后问题)
* [1.8. 栈与队列](#18-栈与队列)
  + [1.8.1. 栈模拟队列](#181-栈模拟队列)
  + [1.8.2. 队列模拟栈](#182-队列模拟栈)
* [1.9. 同步问题](#19-同步问题)
  + [1.9.1. 生产者——消费者模型](#191-生产者消费者模型)
* [1.10. 缓存策略](#110-缓存策略)
  + [1.10.1. LRU](#1101-lru)
  + [1.10.2. LFU](#1102-lfu)
* [1.11. 斐波拉契数列](#111-斐波拉契数列)
  + [1.11.1. 自顶向下](#1111-自顶向下)
  + [1.11.2. 自底向上](#1112-自底向上)
* [1.12. 动态规划](#112-动态规划)
  + [1.12.1. 背包问题](#1121-背包问题)
* [2. 经典题目](#2-经典题目)
* [2.1. 计算数字二进制表示的1的个数](#21-计算数字二进制表示的1的个数)
* [2.2. 数组中除一个数仅出现一次外，其他数均出现两次，找出这个数](#22-数组中除一个数仅出现一次外其他数均出现两次找出这个数)
* [2.3. 长为n的数组中所有元素在[1, n]之间，元素最多出现两次，找出[1, n]中没出现的元素](#23-长为n的数组中所有元素在1-n之间元素最多出现两次找出1-n中没出现的元素)
* [2.4. 不用+和-计算两个数的和](#24-不用和-计算两个数的和)
* [2.5. 反转二叉树](#25-反转二叉树)
* [2.6. 把数组中所有0移动到尾部，数组元素保持相对顺序不变](#26-把数组中所有0移动到尾部数组元素保持相对顺序不变)
* [2.7. 定义一次move是将长为n的数组中n-1个数均加1，最少多少个move可以使得数组中所有元素相等](#27-定义一次move是将长为n的数组中n-1个数均加1最少多少个move可以使得数组中所有元素相等)
* [2.8. 找出长为n的数组中出现次数大于⌊ n/2 ⌋的元素](#28-找出长为n的数组中出现次数大于⌊-n2-⌋的元素)
* [2.9. 找出长为n的数组中出现次数大于⌊ n/k ⌋的元素](#29-找出长为n的数组中出现次数大于⌊-nk-⌋的元素)
* [2.10. 十进制转换为7进制](#210-十进制转换为7进制)
* [2.11. 指定长度二进制数，指定1的个数的所有可能情况](#211-指定长度二进制数指定1的个数的所有可能情况)
* [2.12. 有序数组转二叉搜索树](#212-有序数组转二叉搜索树)
* [2.13. 最大连续子序列和](#213-最大连续子序列和)
  + [2.13.1. 最小值为0](#2131-最小值为0)
  + [2.13.2. 最小值为负数](#2132-最小值为负数)
* [2.14. 最大连续子序列积](#214-最大连续子序列积)
* [2.15. 判断一个数是否为2的幂次](#215-判断一个数是否为2的幂次)
* [2.16. 判断一个数是否为4的幂次](#216-判断一个数是否为4的幂次)
* [2.17. 最近公共祖先问题](#217-最近公共祖先问题)
  + [2.17.1. 二叉树的最近公共祖先](#2171-二叉树的最近公共祖先)
  + [2.17.2. 二叉搜索树的最近公共祖先](#2172-二叉搜索树的最近公共祖先)
* [2.18. 挑选数组中不连续的元素，使其和最大](#218-挑选数组中不连续的元素使其和最大)
* [2.19. 镜像二叉树](#219-镜像二叉树)
* [2.20. 判断二叉树是否平衡](#220-判断二叉树是否平衡)
* [2.21. 二分法求平方根](#221-二分法求平方根)
* [2.22. n!求末尾0的个数](#222-n求末尾0的个数)
* [2.23. 有序数组去重](#223-有序数组去重)
* [2.24. 链表判断环](#224-链表判断环)
* [2.25. 有环链表获取环起始结点](#225-有环链表获取环起始结点)
* [2.26. 判断是否为回文数](#226-判断是否为回文数)
* [2.27. 是否存在二叉树根到叶子节点路径和为指定值](#227-是否存在二叉树根到叶子节点路径和为指定值)
* [2.28. 找出字符串中所有变位词](#228-找出字符串中所有变位词)
* [2.29. 判断两个字符串是否为同构字符串](#229-判断两个字符串是否为同构字符串)
* [2.30. 判断一个数是否为其全部约数的和](#230-判断一个数是否为其全部约数的和)
* [2.31. 打乱数组元素](#231-打乱数组元素)
* [2.32. 生成不重复随机序列](#232-生成不重复随机序列)
* [2.33. 使用随机生成0到5的函数，实现随机生成0到7的函数](#233-使用随机生成0到5的函数实现随机生成0到7的函数)
* [2.34. 计算数组A、B、C、D中有多少种(i, j, k, l)使得A[i] + B[j] + C[k] + D[l] = 0](#234-计算数组abcd中有多少种i-j-k-l使得ai--bj--ck--dl--0)
* [2.35. 计算给出的多个时间中，最小的时间差](#235-计算给出的多个时间中最小的时间差)
* [2.36. 计算[0, 10^n)中各位数字不同的数的个数](#236-计算\0-10%5En中各位数字不同的数的个数)
* [2.37. 数组中仅有两个数仅出现一次，其余的数均出现两次，找出这两个数](#237-数组中仅有两个数仅出现一次其余的数均出现两次找出这两个数)
* [2.38. 找出递增数列中第n位的数字](#238-找出递增数列中第n位的数字)
* [2.39. 计算小于n的质数的个数](#239-计算小于n的质数的个数)
* [2.40. 由部分（可重复）给定数组元素相加得到指定值的情况数](#240-由部分可重复给定数组元素相加得到指定值的情况数)
* [2.41. 在有序矩阵中寻找第k大的数](#241-在有序矩阵中寻找第k大的数)
* [2.42. 求n对括号的所有排列情况](#242-求n对括号的所有排列情况)
* [2.43. 给定字符串数组，求最大不相交的两个字符串的长度积](#243-给定字符串数组求最大不相交的两个字符串的长度积)
* [2.44. 在由[1, n]组成的长为n + 1的数组中，寻找重复的那个数](#244-在由1-n组成的长为n--1的数组中寻找重复的那个数)
* [2.45. 求给定数组的所有排列情况](#245-求给定数组的所有排列情况)
* [2.46. 求给定（元素可重复）数组的所有排列情况](#246-求给定元素可重复数组的所有排列情况)
* [2.47. 最长回文子序列](#247-最长回文子序列)
* [2.48. 将[1, n]的数按字典排序](#248-将1-n的数按字典排序)
* [2.49. 二叉搜索树迭代器](#249-二叉搜索树迭代器)
* [2.50. 由[1, n]组成的二叉搜索树的个数](#250-由1-n组成的二叉搜索树的个数)
* [2.51. 格雷码](#251-格雷码)
* [2.52. 从m x n棋盘左上角走到右下角的路径总数](#252-从m-x-n棋盘左上角走到右下角的路径总数)
* [2.53. 求集合的所有子集](#253-求集合的所有子集)
* [2.54. 求（元素可重复）集合的所有子集](#254-求元素可重复集合的所有子集)
* [2.55. 判断数组arr中是否存在arr[i] < arr[j] < arr[k]](#255-判断数组arr中是否存在arri-lt-arrj-lt-arrk)
* [2.56. 查找数组中第k大的数](#256-查找数组中第k大的数)
* [2.57. 链表删除倒数第n个结点](#257-链表删除倒数第n个结点)
* [2.58. 判断数组是否可以拼接为正方形](#258-判断数组是否可以拼接为正方形)
* [2.59. 快速pow()](#259-快速pow)
* [2.60. 判断数组是否可以分为相等的两部分](#260-判断数组是否可以分为相等的两部分)
* [2.61. 在矩阵中查找指定值](#261-在矩阵中查找指定值)
* [2.62. 将矩阵顺时针旋转90度](#262-将矩阵顺时针旋转90度)
* [2.63. 最长递增子序列](#263-最长递增子序列)
* [2.64. 求二维矩阵由左上角到右下角的最小和](#264-求二维矩阵由左上角到右下角的最小和)
* [2.65. 将由0、1、2组成的数组排序](#265-将由012组成的数组排序)
* [2.66. 求第n位仅由指定因数组成的数字](#266-求第n位仅由指定因数组成的数字)
* [2.67. 数组中查找极大值](#267-数组中查找极大值)
* [2.68. 判断n能否由平方数的和组成](#268-判断n能否由平方数的和组成)
* [2.69. 删除二叉搜索树中的结点](#269-删除二叉搜索树中的结点)
* [2.70. 检查字符串是否是合法的二叉树前序遍历结果](#270-检查字符串是否是合法的二叉树前序遍历结果)
* [2.71. 中缀表达式转后缀表达式](#271-中缀表达式转后缀表达式)
* [2.72. 排列的下一种情况](#272-排列的下一种情况)
* [2.73. 求旋转有序数组的最小值](#273-求旋转有序数组的最小值)
* [2.74. 在旋转有序数组中查找指定值](#274-在旋转有序数组中查找指定值)
* [2.75. 在旋转（元素可重复）有序数组中查找指定值](#275-在旋转元素可重复有序数组中查找指定值)
* [2.76. 有序数组中查找特定元素的第一个和最后一个索引](#276-有序数组中查找特定元素的第一个和最后一个索引)
* [2.77. 有序数组中查找指定值的插入位置](#277-有序数组中查找指定值的插入位置)
* [2.78. 实现pow(x, n)](#278-实现powx-n)
* [2.79. 第k个排列](#279-第k个排列)
* [2.80. 链表向右循环移动k个位置](#280-链表向右循环移动k个位置)
* [2.81. 先序遍历和中序遍历构建二叉树](#281-先序遍历和中序遍历构建二叉树)
* [2.82. 中序遍历和后序遍历构建二叉树](#282-中序遍历和后序遍历构建二叉树)
* [2.83. 求杨辉三角第k行的值](#283-求杨辉三角第k行的值)
* [2.84. 克隆图](#284-克隆图)
* [2.85. 查找数组中的极大值元素](#285-查找数组中的极大值元素)
* [2.86. 将数组合并拼接为最大的数](#286-将数组合并拼接为最大的数)
* [2.87. 计算完全二叉树的结点数](#287-计算完全二叉树的结点数)
* [2.88. 加减乘除（无括号）计算器](#288-加减乘除无括号计算器)
* [2.89. 加减（带括号）计算器](#289-加减带括号计算器)
* [2.90. 二叉搜索树中第k小的数](#290-二叉搜索树中第k小的数)
* [2.91. 求（元素可更新）数组指定区间的和](#291-求元素可更新数组指定区间的和)
* [2.92. 求数组中任意两个元素异或的最大值](#292-求数组中任意两个元素异或的最大值)
* [2.93. 二叉搜索树的序列化和反序列化](#293-二叉搜索树的序列化和反序列化)
* [2.94. 轮流取数判断先取者是否必胜](#294-轮流取数判断先取者是否必胜)
* [2.95. 一串数加上分别加上加号或减号使其结果等于指定值](#295-一串数加上分别加上加号或减号使其结果等于指定值)
* [2.96. 有序数组中除一个数仅出现一次外，其他数均出现两次，求这个数](#296-有序数组中除一个数仅出现一次外其他数均出现两次求这个数)
* [2.97. 最长任意元素差最大为1的子序列](#297-最长任意元素差最大为1的子序列)
* [2.98. 数组中寻找任意三个元素的最大积](#298-数组中寻找任意三个元素的最大积)
* [2.99. 生成结点个数为n的所有二叉搜索树](#299-生成结点个数为n的所有二叉搜索树)
* [2.100. 数组中除一个数仅出现一次外，其他数均出现三次，求这个数](#2100-数组中除一个数仅出现一次外其他数均出现三次求这个数)
* [2.101. 将一个数拆分为多个数的和，并使这些数的积最大](#2101-将一个数拆分为多个数的和并使这些数的积最大)
* [2.102. 轮流取数先取者是否必胜](#2102-轮流取数先取者是否必胜)
* [2.103. 区间调度问题](#2103-区间调度问题)
  + [2.103.1. 最多区间调度](#21031-最多区间调度)
* [2.104. 寻找数组中是否存在nums[i]和nums[j]差的绝对值最大为t，i和j差的绝对值最大为k](#2104-寻找数组中是否存在numsi和numsj差的绝对值最大为ti和j差的绝对值最大为k)
* [2.105. 猜数游戏，猜错需要付钱，求确保猜中的最小代价](#2105-猜数游戏猜错需要付钱求确保猜中的最小代价)
* [2.106. 寻找数组中是否存在i < j < k使得ai < ak < aj](#2106-寻找数组中是否存在i--j--k使得ai--ak--aj)
* [2.107. 矩阵上从某一点经过最多N次移动出界外的路径总数](#2107-矩阵上从某一点经过最多n次移动出界外的路径总数)
* [2.108. 每次操作删除一个字符，求使得两个字符串相同的最少操作次数](#2108-每次操作删除一个字符求使得两个字符串相同的最少操作次数)
* [2.109. 求字符串的（不同下标）回文字串的数目](#2109-求字符串的不同下标回文字串的数目)
* [2.110. 打印二叉树](#2110-打印二叉树)
* [2.111. 是否可以仅改变一个元素，使得数组为单调递增数组](#2111-是否可以仅改变一个元素使得数组为单调递增数组)
* [2.112. 仅调换一次一个数的两个数字，求最大值](#2112-仅调换一次一个数的两个数字求最大值)
* [2.113. 检查带有通配符的括号序列是否合法](#2113-检查带有通配符的括号序列是否合法)
* [2.114. 两个有序数组的中位数](#2114-两个有序数组的中位数)
* [2.115. 带有.和\*的正则表达式匹配](#2115-带有和的正则表达式匹配)
* [2.116. 合并多个有序链表](#2116-合并多个有序链表)
* [2.117. 最长合法括号字串](#2117-最长合法括号字串)
* [2.118. 带有?和\*的正则表达式匹配](#2118-带有和的正则表达式匹配)
* [2.119. 检查字符串是否是合法数字](#2119-检查字符串是否是合法数字)
* [2.120. 通过插入、删除和替换，最少需要多少次操作可以使得两个字符串相同](#2120-通过插入删除和替换最少需要多少次操作可以使得两个字符串相同)
* [2.121. 柱状图中的最大矩形](#2121-柱状图中的最大矩形)
* [2.122. 矩阵中由1组成的最大矩阵](#2122-矩阵中由1组成的最大矩阵)
* [2.123. 判断字符串是否由给定的两个字符串组合而成](#2123-判断字符串是否由给定的两个字符串组合而成)
* [2.124. 一个字符串有多少个子序列可以组成另一个字符串](#2124-一个字符串有多少个子序列可以组成另一个字符串)
* [2.125. 二叉树上最大（不需要经过根结点）路径和](#2125-二叉树上最大不需要经过根结点路径和)
* [2.126. 求将字符串切分为回文字串需要的最少切分次数](#2126-求将字符串切分为回文字串需要的最少切分次数)
* [2.127. 二维坐标系下一条线上最多多少个点](#2127-二维坐标系下一条线上最多多少个点)
* [2.128. 股票交易求最大收益，最多可以进行k次交易](#2128-股票交易求最大收益最多可以进行k次交易)
* [2.129. 在字符串前插入一些字符使得字符串为回文字符串、](#2129-在字符串前插入一些字符使得字符串为回文字符串)
* [2.130. 计算从1到n中含有数字1的数的个数](#2130-计算从1到n中含有数字1的数的个数)
* [2.131. 移动窗口求每个窗口元素最大值](#2131-移动窗口求每个窗口元素最大值)
* [2.132. 数字转换为英文单词组合](#2132-数字转换为英文单词组合)
* [2.133. 二叉树的序列化和反序列化](#2133-二叉树的序列化和反序列化)
* [2.134. 移除字符串中的重复字符，并使得最终结果字典序最小](#2134-移除字符串中的重复字符并使得最终结果字典序最小)
* [2.135. 信封俄罗斯套娃](#2135-信封俄罗斯套娃)
* [2.136. 小矩阵是否能完整拼接为大矩阵](#2136-小矩阵是否能完整拼接为大矩阵)
* [2.137. 将数组划分为m份，使得每份和的最大值最小](#2137-将数组划分为m份使得每份和的最大值最小)
* [2.138. 在1到n中寻找字典序第k小的数](#2138-在1到n中寻找字典序第k小的数)
* [2.139. 滑动窗口中间值](#2139-滑动窗口中间值)
* [2.140. 多个有序数组，每个数组挑选至少一个数，求能够组成的最小的数组范围](#2140-多个有序数组每个数组挑选至少一个数求能够组成的最小的数组范围)
* [2.141. 点游戏](#2141-点游戏)
* [2.142. 数组中寻找三个长为k的不重叠子数组，使得其和最大](#2142-数组中寻找三个长为k的不重叠子数组使得其和最大)
* [2.143. 快速替换10亿条标题中的5万个敏感词](#2143-快速替换10亿条标题中的5万个敏感词)
* [2.144. 轮流取数的必胜策略](#2144-轮流取数的必胜策略)
* [2.145. 最大公约数（辗转相除法）](#2145-最大公约数辗转相除法)
* [2.146. 最小公倍数](#2146-最小公倍数)

## 1. 数据结构

### 1.1. 单例模式

public class Singleton {  
 private volatile static Singleton instance;  
  
 private Singleton() {}  
  
 public static Singleton getInstance() {  
 if (instance == null) {  
 synchronized (Singleton.class) {  
 if (instance == null) {  
 instance = new Singleton();  
 }  
 }  
 }  
 return instance;  
 }  
}

### 1.2. 排序

#### 1.2.1. 快速排序partition过程

##### 2.1.1. 方法一

def partition(array, low, high):  
 pivot = array[low]  
 while low < high:  
 while low < high and array[high] >= pivot:  
 high -= 1  
 array[low] = array[high]  
 while low < high and array[low] <= pivot:  
 low += 1  
 array[high] = array[low]  
 array[low] = pivot  
 return low

##### 2.1.2. 方法二

def partition2(array, low, high):  
 def swap(array, i, j):  
 array[i], array[j] = array[j], array[i]  
  
 pivot = array[high]  
 current = low  
 for i in range(low, high):  
 if array[i] < pivot:  
 swap(array, i, current)  
 current += 1  
 swap(array, high, current)  
 return current

##### 2.1.3. 方法三

def partition3(array, low, high):  
 def swap(array, i, j):  
 array[i], array[j] = array[j], array[i]  
  
 start = low  
 pivot = array[low]  
 low += 1  
 while low <= high:  
 while low <= high and array[low] <= pivot:  
 low += 1  
 while low <= high and array[high] >= pivot:  
 high -= 1  
 if low < high:  
 swap(array, low, high)  
 swap(array, start, high)  
 return high

#### 1.2.2. 快速排序

##### 2.2.1. 递归

def quick\_sort(array, low, high):  
 if low >= high:  
 return  
 mid = partition(array, low, high)  
 quick\_sort(array, low, mid - 1)  
 quick\_sort(array, mid + 1, high)

#### 1.2.3. 非递归

def quick\_sort(array, low, high):  
 stack = []  
 stack.append(low)  
 stack.append(high)  
 while stack:  
 high = stack.pop()  
 low = stack.pop()  
 if low >= high:  
 continue  
 mid = partition(array, low, high)  
 stack.append(low)  
 stack.append(mid - 1)  
 stack.append(mid + 1)  
 stack.append(high)

#### 1.2.4. 插入排序

def insertion\_sort(array):  
 for i in range(1, len(array)):  
 temp = array[i]  
 j = i - 1  
 while j >= 0 and temp < array[j]:  
 array[j + 1] = array[j]  
 j -= 1  
 array[j + 1] = temp

#### 1.2.5. 冒泡排序

def bubble\_sort(array):  
 def swap(array, i, j):  
 array[i], array[j] = array[j], array[i]  
  
 for j in range(len(array) - 1, 0, -1):  
 flag = True  
 for i in range(0, j):  
 if array[i] > array[i + 1]:  
 swap(array, i, i + 1)  
 flag = False  
 if flag:  
 break

#### 1.2.6. 选择排序

def selection\_sort(array):  
 def swap(array, i, j):  
 array[i], array[j] = array[j], array[i]  
  
 for i in range(0, len(array) - 1):  
 min\_index = i  
 for j in range(i + 1, len(array)):  
 if array[j] < array[min\_index]:  
 min\_index = j  
 swap(array, i, min\_index)

#### 1.2.7. 归并排序

##### 2.6.1. 递归

def merge(array, temp\_array, low, mid, high):  
 start1 = low  
 end1 = mid  
 start2 = mid + 1  
 end2 = high  
 k = low  
 while start1 <= end1 and start2 <= end2:  
 if array[start1] < array[start2]:  
 temp\_array[k] = array[start1]  
 start1 += 1  
 else:  
 temp\_array[k] = array[start2]  
 start2 += 1  
 k += 1  
 while start1 <= end1:  
 temp\_array[k] = array[start1]  
 start1 += 1  
 k += 1  
 while start2 <= end2:  
 temp\_array[k] = array[start2]  
 start2 += 1  
 k += 1  
 for k in range(low, high + 1):  
 array[k] = temp\_array[k]  
  
  
def merge\_sort\_real(array, temp\_array, low, high):  
 if low >= high:  
 return  
 mid = (low + high) // 2  
 merge\_sort\_real(array, temp\_array, low, mid)  
 merge\_sort\_real(array, temp\_array, mid + 1, high)  
 merge(array, temp\_array, low, mid, high)  
  
  
def merge\_sort(array):  
 merge\_sort\_real(array, [0] \* len(array), 0, len(array) - 1)

##### 2.6.2. 非递归

def merge\_sort(array):  
 length = len(array)  
 temp\_array = [0] \* length  
 block = 1  
 while block < length \* 2:  
 for start in range(0, length, 2 \* block):  
 low = start  
 mid = (start + block) if (start + block) < length else length  
 high = (start + 2 \* block) if (start + 2 \* block) < length else length  
 start1 = low  
 end1 = mid  
 start2 = mid  
 end2 = high  
 k = low  
 while start1 < end1 and start2 < end2:  
 if array[start1] < array[start2]:  
 temp\_array[k] = array[start1]  
 start1 += 1  
 else:  
 temp\_array[k] = array[start2]  
 start2 += 1  
 k += 1  
 while start1 < end1:  
 temp\_array[k] = array[start1]  
 start1 += 1  
 k += 1  
 while start2 < end2:  
 temp\_array[k] = array[start2]  
 start2 += 1  
 k += 1  
 array, temp\_array = temp\_array, array  
 block \*= 2

#### 1.2.8. 堆排序

def heap\_sort(array):  
 def swap(array, i, j):  
 array[i], array[j] = array[j], array[i]  
  
 def shift\_down(start, end):  
 root = start  
 while True:  
 child = 2 \* root + 1  
 if child > end:  
 break  
 if child + 1 <= end and array[child] < array[child + 1]:  
 child += 1  
 if array[root] < array[child]:  
 swap(array, root, child)  
 root = child  
 else:  
 break  
  
 def build\_heap(array):  
 length = len(array)  
 for start in range((length - 1) // 2, -1, -1):  
 shift\_down(start, length - 1)  
  
 build\_heap(array)  
 for end in range(len(array) - 1, 0, -1):  
 swap(array, 0, end)  
 shift\_down(0, end - 1)

### 1.3. 链表

#### 1.3.1. 反转链表（头插法）

def reverse\_list(head):  
 if not head:  
 return head  
 new\_head = head  
 while head.next:  
 current = head.next  
 head.next = head.next.next  
 current.next = new\_head  
 new\_head = current  
 return new\_head

#### 1.3.2. 单链表去重

public ListNode deleteDuplicates(ListNode head) {  
 ListNode p = head;  
 while (p != null) {  
 if (p.next != null && p.next.val == p.val) {  
 p.next = p.next.next;  
 } else {  
 p = p.next;  
 }  
 }  
 return head;  
}

#### 1.3.3. 合并两个链表

##### 3.3.1. 不创建临时结点

public ListNode mergeTwoLists(ListNode l1, ListNode l2) {  
 if (l1 == null) {  
 return l2;  
 }  
 if (l2 == null) {  
 return l1;  
 }  
 if (l1.val > l2.val) {  
 ListNode temp = l1;  
 l1 = l2;  
 l2 = temp;  
 }  
 ListNode p = l1;  
 while (p != null) {  
 while (l2 != null && (p.next == null || l2.val < p.next.val)) {  
 ListNode temp = l2;  
 l2 = l2.next;  
 temp.next = p.next;  
 p.next = temp;  
 p = p.next;  
 }  
 p = p.next;  
 }  
 return l1;  
}

##### 3.3.2. 创建临时结点

public ListNode mergeTwoLists(ListNode l1, ListNode l2) {  
 ListNode fakeHead = new ListNode(0);  
 ListNode tail = fakeHead;  
 while (l1 != null && l2 != null) {  
 if (l1.val < l2.val) {  
 tail.next = l1;  
 l1 = l1.next;  
 } else {  
 tail.next = l2;  
 l2 = l2.next;  
 }  
 tail = tail.next;  
 }  
 tail.next = l1 == null ? l2 : l1;  
 return fakeHead.next;  
}

#### 1.3.4. 链表排序

def sortList(self, head):  
 """  
 :type head: ListNode  
 :rtype: ListNode  
 """  
 if not head or not head.next:  
 return head  
 slow = head  
 fast = head  
 prev = None  
 while fast and fast.next:  
 prev = slow  
 slow = slow.next  
 fast = fast.next.next  
 prev.next = None  
 a = sortList(head)  
 b = sortList(slow)  
 fake\_head = ListNode(0)  
 temp = fake\_head  
 while a and b:  
 if a.val < b.val:  
 temp.next = a  
 a = a.next  
 else:  
 temp.next = b  
 b = b.next  
 temp = temp.next  
 if a:  
 temp.next = a  
 else:  
 temp.next = b  
 return fake\_head.next

#### 1.3.5. 检查回文链表

def isPalindrome(head):  
 """  
 :type head: ListNode  
 :rtype: bool  
 """  
 if not head or not head.next:  
 return True  
 slow = head  
 fast = head  
 while fast and fast.next:  
 slow = slow.next  
 fast = fast.next.next  
 temp = None  
 while slow:  
 next = slow.next  
 slow.next = temp  
 temp = slow  
 slow = next  
 slow = temp  
 while head and slow:  
 if head.val != slow.val:  
 return False  
 head = head.next  
 slow = slow.next  
 return True

#### 1.3.6. 链表等概率随机抽取元素

public int getRandom(ListNode head) {  
 ListNode node = head;  
 int result = -1;  
 int count = 1;  
 Random random = new Random();  
 while (node != null) {  
 if (random.nextInt(count) == 0) {  
 result = node.val;  
 }  
 count++;  
 node = node.next;  
 }  
 return result;  
}

#### 1.3.7. 链表按奇数序号和偶数序号重排

def oddEvenList(head):  
 """  
 :type head: ListNode  
 :rtype: ListNode  
 """  
 if not head:  
 return None  
 odd = head  
 even = head.next  
 even\_head = even  
 while even and even.next:  
 odd.next = odd.next.next  
 even.next = even.next.next  
 odd = odd.next  
 even = even.next  
 odd.next = even\_head  
 return head

### 1.4. 二叉树遍历

#### 1.4.1. 二叉树前序遍历

##### 4.1.1. 递归

def pre\_order(root):  
 if root:  
 print(root.val)  
 pre\_order(root.left)  
 pre\_order(root.right)

##### 4.1.2. 非递归

def pre\_order(root):  
 stack = []  
 while root or stack:  
 if root:  
 print(root.val)  
 stack.append(root)  
 root = root.left  
 else:  
 node = stack.pop()  
 root = node.right

#### 1.4.2. 二叉树中序遍历

##### 4.2.1. 递归

def in\_order(root):  
 if root:  
 in\_order(root.left)  
 print(root.val)  
 in\_order(root.right)

##### 4.2.2. 非递归

def in\_order(root):  
 stack = []  
 while root or stack:  
 if root:  
 stack.append(root)  
 root = root.left  
 else:  
 node = stack.pop()  
 print(node.val)  
 root = node.right

#### 1.4.3. 二叉树后序遍历

##### 4.3.1. 递归

def post\_order(root):  
 if root:  
 post\_order(root.left)  
 post\_order(root.right)  
 print(root.val)

##### 4.3.2. 非递归

def post\_order(root):  
 stack = []  
 result = []  
 node = root  
 while node or stack:  
 if node:  
 stack.append(node)  
 result.append(node.val)  
 node = node.right  
 else:  
 node = stack.pop()  
 node = node.left  
 result.reverse()  
 return result

#### 1.4.4. 二叉树层次遍历

def level\_order(root):  
 if not root:  
 return  
 queue = collections.deque()  
 queue.append(root)  
 while queue:  
 size = len(queue)  
 for i in range(size):  
 node = queue.popleft()  
 print(node.val)  
 if node.left:  
 queue.append(node.left)  
 if node.right:  
 queue.append(node.right)

### 1.5. 图

#### 1.5.1. Dijkstra算法

##### 5.1.1. 朴素

def dijkstra(start, end, graph, n):  
 """  
 :param start: int, [1,n]  
 :param end: int, [1,n]  
 :param graph: dict, {from: (to, distance), from: (to, distance)...}  
 :param n: int  
 :return: (int, list), minimum distance, path  
 """  
 d = [sys.maxsize] \* (n + 1)  
 previous = [None] \* (n + 1)  
 d[start] = 0  
 candidates = set(i for i in range(1, n + 1))  
 while candidates:  
 min\_d = sys.maxsize  
 u = -1  
 for candidate in candidates:  
 if min\_d > d[candidate]:  
 min\_d = d[candidate]  
 u = candidate  
 if u == end:  
 return d[u], previous  
 candidates.remove(u)  
 for v, dist in graph[u]:  
 if d[v] > d[u] + dist:  
 d[v] = d[u] + dist  
 previous[v] = u

##### 5.1.2. 堆优化

def dijkstra(start, end, graph, n):  
 """  
 :param start: int, [1,n]  
 :param end: int, [1,n]  
 :param graph: dict, {from: (to, distance), from: (to, distance)...}  
 :param n: int  
 :return: (int, list), minimum distance, path  
 """  
 d = [sys.maxsize] \* (n + 1)  
 previous = [None] \* (n + 1)  
 d[start] = 0  
 candidates = set(i for i in range(1, n + 1))  
 heap = []  
 heapq.heappush(heap, (0, start))  
 while candidates:  
 distance, u = heapq.heappop()  
 if u == end:  
 return d[u], previous  
 if u not in candidates:  
 continue  
 candidates.remove(u)  
 for v, dist in graph[u]:  
 if d[v] > d[u] + dist:  
 d[v] = d[u] + dist  
 heapq.heappush(heap, (d[v], v))  
 previous[v] = u

#### 1.5.2. Floyd-Warshall算法

def floyd\_warshall(graph, n):  
 INF = 10000  
 dist = [[INF] \* n for \_ in range(n)]  
 path = [[-1] \* n for \_ in range(n)]  
 for i in range(n):  
 for j in range(n):  
 if i == j:  
 dist[i][j] = 0  
 elif graph[i][j] < INF:  
 dist[i][j] = graph[i][j]  
 for k in range(n):  
 for i in range(n):  
 for j in range(n):  
 if dist[i][j] > dist[i][k] + dist[k][j]:  
 dist[i][j] = dist[i][k] + dist[k][j]  
 path[i][j] = k  
 return dist, path

#### 1.5.3. 拓扑排序

##### 5.3.1. DFS

def topological\_sort(graph, n):  
 order = []  
 visiting = [False] \* n  
 visited = [False] \* n  
 for i in range(n):  
 if not visited[i]:  
 if not dfs(graph, i, visiting, visited, order):  
 raise ValueError('cycle')  
 order.reverse()  
 return order  
  
  
def dfs(graph, index, visiting, visited, order):  
 visiting[index] = True  
 visited[index] = True  
 for neighbor in graph[index]:  
 if visiting[neighbor]:  
 return False  
 if not visited[neighbor]:  
 if not dfs(graph, neighbor, visiting, visited, order):  
 return False  
 order.append(index)  
 visiting[index] = False  
 return True

##### 5.3.2. BFS

def topological\_sort(graph, n):  
 in\_degrees = [0] \* n  
 for i in range(n):  
 for index in graph[i]:  
 in\_degrees[index] += 1  
 candidates = set()  
 for i in range(n):  
 if in\_degrees[i] == 0:  
 candidates.add(i)  
 order = []  
 while candidates:  
 index = candidates.pop()  
 order.append(index)  
 for neighbor in graph[index]:  
 in\_degrees[neighbor] -= 1  
 if in\_degrees[neighbor] == 0:  
 candidates.add(neighbor)  
 if len(order) != n:  
 raise ValueError('cycle')  
 return order

### 1.6. 查找

#### 1.6.1. 二分搜索

def binary\_search(array, key):  
 low = 0  
 high = len(array) - 1  
 while low <= high:  
 mid = (low + high) // 2  
 if array[mid] == key:  
 return mid  
 elif array[mid] > key:  
 high = mid - 1  
 else:  
 low = mid + 1  
 return -1

#### 1.6.2. 并查集

class UnionFind:  
 def \_\_init\_\_(self, n):  
 self.count = n  
 self.parent = [i for i in range(n)]  
 self.rank = [0] \* n  
  
 def find(self, num):  
 if num != self.parent[num]:  
 self.parent[num] = self.find(self.parent[num])  
 return self.parent[num]  
  
 def union(self, a, b):  
 parent\_a = self.find(a)  
 parent\_b = self.find(b)  
 if parent\_a == parent\_b:  
 return  
 if self.rank[parent\_a] > self.rank[parent\_b]:  
 self.parent[parent\_b] = parent\_a  
 else:  
 self.parent[parent\_a] = parent\_b  
 if self.rank[parent\_a] == self.rank[parent\_b]:  
 self.rank[parent\_b] += 1  
 self.count -= 1

#### 1.6.3. Trie树

public class Trie {  
 TrieNode root;  
  
 class TrieNode {  
 boolean isWord = false;  
 char val;  
 TrieNode[] children = new TrieNode[26];  
 }  
  
 public Trie() {  
 root = new TrieNode();  
 }  
  
 public void insert(String word) {  
 TrieNode current = root;  
 for (char letter : word.toCharArray()) {  
 if (current.children[letter - 'a'] == null) {  
 current.children[letter - 'a'] = new TrieNode();  
 current.children[letter - 'a'].val = letter;  
 }  
 current = current.children[letter - 'a'];  
 }  
 current.isWord = true;  
 }  
  
 public boolean search(String word) {  
 TrieNode current = root;  
 for (char letter : word.toCharArray()) {  
 if (current.children[letter - 'a'] == null) {  
 return false;  
 }  
 current = current.children[letter - 'a'];  
 }  
 return current.isWord;  
 }  
  
 public boolean startsWith(String prefix) {  
 TrieNode current = root;  
 for (char letter : prefix.toCharArray()) {  
 if (current.children[letter - 'a'] == null) {  
 return false;  
 }  
 current = current.children[letter - 'a'];  
 }  
 return true;  
 }  
}

### 1.7. 回溯

#### 1.7.1. N皇后问题

def n\_queens(n):  
 """  
 :type n: int  
 :rtype: List[List[str]]  
 """  
 board = [['.'] \* n for \_ in range(n)]  
 result = []  
 back\_trace(board, 0, result)  
 return result  
  
  
def back\_trace(board, row\_index, result):  
 if row\_index == len(board):  
 result.append(convert(board))  
 return  
 for j in range(len(board[row\_index])):  
 if is\_valid(board, row\_index, j):  
 board[row\_index][j] = 'Q'  
 back\_trace(board, row\_index + 1, result)  
 board[row\_index][j] = '.'  
  
  
def convert(board):  
 return [''.join(row) for row in board]  
  
  
def is\_valid(board, i, j):  
 for x in range(i):  
 for y in range(len(board[i])):  
 if board[x][y] == 'Q' and (y == j or i - x == j - y or i - x == y - j):  
 return False  
 return True

### 1.8. 栈与队列

#### 1.8.1. 栈模拟队列

public class MyQueue {  
 Stack<Integer> stackA;  
 Stack<Integer> stackB;  
  
 public MyQueue() {  
 stackA = new Stack<>();  
 stackB = new Stack<>();  
 }  
  
 public void push(int x) {  
 stackA.add(x);  
 }  
  
 public int pop() {  
 if (stackB.empty()) {  
 while (!stackA.empty()) {  
 stackB.add(stackA.pop());  
 }  
 }  
 return stackB.pop();  
 }  
  
 public int peek() {  
 if (stackB.empty()) {  
 while (!stackA.empty()) {  
 stackB.add(stackA.pop());  
 }  
 }  
 return stackB.peek();  
 }  
  
 public boolean empty() {  
 return stackA.empty() && stackB.empty();  
 }  
}

#### 1.8.2. 队列模拟栈

public class MyStack {  
 Queue<Integer> queue;  
  
 public MyStack() {  
 queue = new LinkedList<>();  
 }  
  
 public void push(int x) {  
 queue.add(x);  
 for (int i = 0; i < queue.size() - 1; i++) {  
 queue.add(queue.poll());  
 }  
 }  
  
 public int pop() {  
 return queue.poll();  
 }  
  
 public int top() {  
 return queue.peek();  
 }  
  
 public boolean empty() {  
 return queue.isEmpty();  
 }  
}

### 1.9. 同步问题

#### 1.9.1. 生产者——消费者模型

public class Main {  
 public static void main(String[] args) {  
 Queue<Integer> queue = new LinkedList<>();  
 int maxSize = 10;  
 Thread producer1 = new Producer(queue, maxSize, "Producer 1");  
 Thread producer2 = new Producer(queue, maxSize, "Producer 2");  
 Thread consumer1 = new Consumer(queue, "Consumer 1");  
 Thread consumer2 = new Consumer(queue, "Consumer 2");  
  
 producer1.start();  
 consumer1.start();  
 producer2.start();  
 consumer2.start();  
 }  
  
 static class Producer extends Thread {  
 Queue<Integer> queue;  
 int maxSize;  
 Random random;  
  
 public Producer(Queue<Integer> queue, int maxSize, String name) {  
 super(name);  
 this.queue = queue;  
 this.maxSize = maxSize;  
 random = new Random();  
 }  
  
 @Override  
 public void run() {  
 while (true) {  
 try {  
 produce();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
 void produce() throws InterruptedException {  
 synchronized (queue) {  
 while (queue.size() == maxSize) {  
 System.out.println("Queue is full, " + this.getName() + " is waiting");  
 queue.wait();  
 }  
 int item = random.nextInt();  
 queue.add(item);  
 System.out.println(this.getName() + " produced " + item + ", queue size is " + queue.size() + " now");  
 queue.notifyAll();  
 Thread.sleep(new Random().nextInt(1000));  
 }  
 }  
 }  
  
 static class Consumer extends Thread {  
 Queue<Integer> queue;  
 Random random;  
  
 public Consumer(Queue<Integer> queue, String name) {  
 super(name);  
 this.queue = queue;  
 random = new Random();  
 }  
  
 @Override  
 public void run() {  
 while (true) {  
 try {  
 consume();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
 void consume() throws InterruptedException {  
 synchronized (queue) {  
 while (queue.isEmpty()) {  
 System.out.println("Queue is empty, " + this.getName() + " is waiting");  
 queue.wait();  
 }  
 int item = queue.remove();  
 System.out.println(this.getName() + " consumed " + item + ", queue size is " + queue.size() + " now");  
 queue.notifyAll();  
 Thread.sleep(new Random().nextInt(1000));  
 }  
 }  
 }  
}

### 1.10. 缓存策略

#### 1.10.1. LRU

class LRUCache:  
 class Node:  
 def \_\_init\_\_(self, key, value):  
 self.key = key  
 self.val = value  
 self.prev = None  
 self.next = None  
  
 def \_\_init\_\_(self, capacity):  
 """  
 :type capacity: int  
 """  
 self.capacity = capacity  
 self.memory = {}  
 self.head = self.Node(-1, -1)  
 self.tail = self.Node(-1, -1)  
 self.head.next = self.tail  
 self.tail.prev = self.head  
  
 def get(self, key):  
 """  
 :type key: int  
 :rtype: int  
 """  
 if key not in self.memory:  
 return -1  
 node = self.memory[key]  
 self.to\_head(node)  
 return node.val  
  
 def put(self, key, value):  
 """  
 :type key: int  
 :type value: int  
 :rtype: void  
 """  
 if key in self.memory:  
 node = self.memory[key]  
 node.val = value  
 self.to\_head(node)  
 else:  
 if len(self.memory) == self.capacity:  
 del self.memory[self.tail.prev.key]  
 self.remove(self.tail.prev)  
 node = self.Node(key, value)  
 self.memory[key] = node  
 self.add(node)  
  
 def to\_head(self, node):  
 self.remove(node)  
 self.add(node)  
  
 def remove(self, node):  
 node.prev.next = node.next  
 node.next.prev = node.prev  
  
 def add(self, node):  
 node.next = self.head.next  
 self.head.next = node  
 node.prev = self.head  
 node.next.prev = node

#### 1.10.2. LFU

class LFUCache:  
 class Node:  
 def \_\_init\_\_(self, key, value, times):  
 self.key = key  
 self.value = value  
 self.times = times  
 self.prev = None  
 self.next = None  
  
 def \_\_init\_\_(self, capacity):  
 """  
 :type capacity: int  
 """  
 self.init\_capacity = capacity  
 self.capacity = capacity  
 self.memory\_nodes = {}  
 self.memory\_times = {}  
 self.head = self.Node(-1, -1, -1)  
 self.tail = self.Node(-1, -1, -1)  
 self.head.next = self.tail  
 self.tail.prev = self.head  
  
 def get(self, key):  
 """  
 :type key: int  
 :rtype: int  
 """  
 if key not in self.memory\_nodes:  
 return -1  
 self.put(key, self.memory\_nodes[key].value)  
 return self.memory\_nodes[key].value  
  
 def put(self, key, value):  
 """  
 :type key: int  
 :type value: int  
 :rtype: void  
 """  
 if self.init\_capacity == 0:  
 return  
 if key in self.memory\_nodes:  
 node = self.memory\_nodes[key]  
 node.value = value  
  
 if node.times + 1 in self.memory\_times:  
 if self.memory\_times[node.times] == node:  
 if node.next.times == node.times:  
 self.memory\_times[node.times] = node.next  
 else:  
 del self.memory\_times[node.times]  
 node.prev.next = node.next  
 node.next.prev = node.prev  
 node.prev = self.memory\_times[node.times + 1].prev  
 node.next = self.memory\_times[node.times + 1]  
 node.next.prev = node  
 node.prev.next = node  
 self.memory\_times[node.times + 1] = node  
 else:  
 if self.memory\_times[node.times] != node:  
 node.prev.next = node.next  
 node.next.prev = node.prev  
 node.prev = self.memory\_times[node.times].prev  
 node.next = self.memory\_times[node.times]  
 node.next.prev = node  
 node.prev.next = node  
 else:  
 if node.next.times == node.times:  
 self.memory\_times[node.times] = node.next  
 else:  
 del self.memory\_times[node.times]  
 self.memory\_times[node.times + 1] = node  
 node.times += 1  
 else:  
 node = self.Node(key, value, 1)  
 self.memory\_nodes[key] = node  
 if self.capacity == 0:  
 node\_to\_remove = self.tail.prev  
 if self.memory\_times[node\_to\_remove.times] == node\_to\_remove:  
 del self.memory\_times[node\_to\_remove.times]  
 node\_to\_remove.prev.next = node\_to\_remove.next  
 node\_to\_remove.next.prev = node\_to\_remove.prev  
 del self.memory\_nodes[node\_to\_remove.key]  
 self.capacity += 1  
 if 1 in self.memory\_times:  
 node.prev = self.memory\_times[1].prev  
 node.next = self.memory\_times[1]  
  
 else:  
 node.prev = self.tail.prev  
 node.next = self.tail  
 node.next.prev = node  
 node.prev.next = node  
 self.memory\_times[1] = node  
 self.capacity -= 1

### 1.11. 斐波拉契数列

#### 1.11.1. 自顶向下

def fibonacci(n):  
 if n == 1 or n == 2:  
 return 1  
 return fibonacci(n - 1) + fibonacci(n - 2)

#### 1.11.2. 自底向上

def fibonacci(n):  
 a = 1  
 b = 1  
 n -= 2  
 while n > 0:  
 temp = a + b  
 a = b  
 b = temp  
 n -= 1  
 return b

### 1.12. 动态规划

#### 1.12.1. 背包问题

##### 01背包

###### 求最大价值

def knapsack(max\_weight, weights, values):  
 n = len(weights)  
 dp = [[0] \* (max\_weight + 1) for \_ in range(n + 1)]  
 for i in range(1, n + 1):  
 for j in range(1, max\_weight + 1):  
 dp[i][j] = dp[i - 1][j]  
 if weights[i - 1] <= j:  
 dp[i][j] = max(dp[i][j], dp[i - 1][j - weights[i - 1]] + values[i - 1])  
 return dp[n][max\_weight]  
  
def knapsack(max\_weight, weights, values):  
 n = len(weights)  
 dp = [0] \* (max\_weight + 1)  
 for i in range(1, n + 1):  
 for j in range(max\_weight, 0, -1):  
 if weights[i - 1] <= j:  
 dp[j] = max(dp[j], dp[j - weights[i - 1]] + values[i - 1])  
 return dp[max\_weight]

if \_\_name\_\_ == '\_\_main\_\_':  
 weights = [10, 20, 30]  
 values = [60, 100, 120]  
 max\_weight = 50  
 print(knapsack(max\_weight, weights, values))

###### 输出价值最大时的方案

def knapsack(max\_weight, weights, values):  
 n = len(weights)  
 dp = [[0] \* (max\_weight + 1) for \_ in range(n + 1)]  
 for i in range(1, n + 1):  
 for j in range(1, max\_weight + 1):  
 dp[i][j] = dp[i - 1][j]  
 if weights[i - 1] <= j:  
 dp[i][j] = max(dp[i][j], dp[i - 1][j - weights[i - 1]] + values[i - 1])  
 j = max\_weight  
 result = []  
 for i in range(n, 0, -1):  
 if dp[i][j] > dp[i - 1][j]:  
 result.append(i - 1)  
 j -= weights[i - 1]  
 result.reverse()  
 return result

if \_\_name\_\_ == '\_\_main\_\_':  
 weights = [2, 2, 6, 5, 4]  
 values = [6, 3, 5, 4, 6]  
 max\_weight = 10  
 print(knapsack(max\_weight, weights, values))

## 2. 经典题目

### 2.1. 计算数字二进制表示的1的个数

[Leetcode 461. Hamming Distance](https://leetcode.com/problems/hamming-distance/#/description)

def count\_1s(num):  
 count = 0  
 while num:  
 num &= (num - 1)  
 count += 1  
 return count

### 2.2. 数组中除一个数仅出现一次外，其他数均出现两次，找出这个数

[Leetcode 136. Single Number](https://leetcode.com/problems/single-number/#/description)

def single\_number(nums):  
 result = 0  
 for num in nums:  
 result ^= num  
 return result

### 2.3. 长为n的数组中所有元素在[1, n]之间，元素最多出现两次，找出[1, n]中没出现的元素

[Leetcode 448. Find All Numbers Disappeared in an Array](https://leetcode.com/problems/find-all-numbers-disappeared-in-an-array/#/description)

def find\_disappeared\_numbers(nums):  
 """  
 :type nums: List[int]  
 :rtype: List[int]  
 """  
 for num in nums:  
 index = num if num > 0 else -num  
 index -= 1  
 if nums[index] > 0:  
 nums[index] = -nums[index]  
 result = []  
 for (index, num) in enumerate(nums):  
 if num > 0:  
 result.append(index + 1)  
 return result

### 2.4. 不用+和-计算两个数的和

[Leetcode 371. Sum of Two Integers](https://leetcode.com/problems/sum-of-two-integers/#/description)

public int getSum(int a, int b) {  
 int sum = a ^ b;  
 int part = a & b;  
 while (part != 0) {  
 int theA = sum;  
 int theB = part << 1;  
 sum = theA ^ theB;  
 part = theA & theB;  
 }  
 return sum;  
}

### 2.5. 反转二叉树

[Leetcode 226. Invert Binary Tree](https://leetcode.com/problems/invert-binary-tree/#/description)

def invertTree(root):  
 """  
 :type root: TreeNode  
 :rtype: TreeNode  
 """  
 if root is None:  
 return  
 self.invertTree(root.left)  
 self.invertTree(root.right)  
 temp = root.left  
 root.left = root.right  
 root.right = temp  
 return root

### 2.6. 把数组中所有0移动到尾部，数组元素保持相对顺序不变

[Leetcode 283. Move Zeroes](https://leetcode.com/problems/move-zeroes/#/description)

void moveZeroes(int[] nums) {  
 int currentIndex = 0;  
 for (int num : nums) {  
 if (num != 0) nums[currentIndex] = num;  
 currentIndex++;  
 }  
 while (currentIndex < nums.length) {  
 nums[currentIndex] = 0;  
 currentIndex++;  
 }  
}

### 2.7. 定义一次move是将长为n的数组中n-1个数均加1，最少多少个move可以使得数组中所有元素相等

[Leetcode 453. Minimum Moves to Equal Array Elements](https://leetcode.com/problems/minimum-moves-to-equal-array-elements/#/description)

def minMoves(nums):  
 """  
 :type nums: List[int]  
 :rtype: int  
 """  
 return sum(nums) - min(nums) \* len(nums)

### 2.8. 找出长为n的数组中出现次数大于⌊ n/2 ⌋的元素

[Leetcode 169. Majority Element](https://leetcode.com/problems/majority-element/#/description)

def majorityElement(nums):  
 """  
 :type nums: List[int]  
 :rtype: int  
 """  
 majority = None  
 count = 0  
 for num in nums:  
 if count == 0:  
 majority = num  
 if majority == num:  
 count += 1  
 else:  
 count -= 1  
 return majority

### 2.9. 找出长为n的数组中出现次数大于⌊ n/k ⌋的元素

[Leetcode 229. Majority Element II](https://leetcode.com/problems/majority-element-ii/description/)

def majority\_element(nums, k):  
 counters = {}  
 for num in nums:  
 if num in counters:  
 counters[num] += 1  
 elif len(counters.keys()) < k - 1:  
 counters[num] = 1  
 else:  
 for key, count in counters.items():  
 if count == 1:  
 del counters[key]  
 else:  
 counters[key] -= 1  
 for num in counters.keys():  
 counters[num] = 0  
 for num in nums:  
 if num in counters:  
 counters[num] += 1  
 target = len(nums) // k  
 result = []  
 for num, count in counters.items():  
 if count > target:  
 result.append(num)  
 return result

### 2.10. 十进制转换为7进制

[Leetcode 504. Base 7](https://leetcode.com/problems/base-7/#/description)

public String convertToBase7(int num) {  
 if (num == 0) {  
 return "0";  
 }  
 int sign = num >= 0 ? 1 : -1;  
 num = Math.abs(num);  
 StringBuilder builder = new StringBuilder();  
 while (num != 0) {  
 builder.append(num % 7);  
 num /= 7;  
 }  
 if (sign < 0) {  
 builder.append('-');  
 }  
 return builder.reverse().toString();  
}

### 2.11. 指定长度二进制数，指定1的个数的所有可能情况

[Leetcode 401. Binary Watch](https://leetcode.com/problems/binary-watch/#/description)

def get\_combination(length, num\_ones):  
 def combination(length, num\_ones, base, result):  
 if num\_ones == 0:  
 result.append(base)  
 return  
 if length == 0 or length < num\_ones:  
 return  
 combination(length - 1, num\_ones - 1, base + (1 << (length - 1)), result)  
 combination(length - 1, num\_ones, base, result)  
  
 result = []  
 combination(length, num\_ones, 0, result)  
 return result

### 2.12. 有序数组转二叉搜索树

[Leetcode 108. Convert Sorted Array to Binary Search Tree](https://leetcode.com/problems/convert-sorted-array-to-binary-search-tree/#/description)

TreeNode toBST(int[] nums, int start, int end) {  
 if (start > end) {  
 return null;  
 }  
 int mid = (start + end) / 2;  
 TreeNode node = new TreeNode(nums[mid]);  
 node.left = toBST(nums, start, mid - 1);  
 node.right = toBST(nums, mid + 1, end);  
 return node;  
}  
  
public TreeNode sortedArrayToBST(int[] nums) {  
 if (nums.length == 0) {  
 return null;  
 }  
 return toBST(nums, 0, nums.length - 1);  
}

### 2.13. 最大连续子序列和

[Leetcode 121. Best Time to Buy and Sell Stock](https://leetcode.com/problems/best-time-to-buy-and-sell-stock/#/description)

#### 2.13.1. 最小值为0

def max\_subarray(nums):  
 max\_ending\_here = 0  
 max\_so\_far = 0  
 for x in nums:  
 max\_ending\_here = max(0, max\_ending\_here + x)  
 max\_so\_far = max(max\_so\_far, max\_ending\_here)  
 return max\_so\_far

#### 2.13.2. 最小值为负数

def max\_subarray(nums):  
 max\_ending\_here = nums[0]  
 max\_so\_far = nums[0]  
 for x in nums[1:]:  
 max\_ending\_here = max(x, max\_ending\_here + x)  
 max\_so\_far = max(max\_so\_far, max\_ending\_here)  
 return max\_so\_far

### 2.14. 最大连续子序列积

[Leetcode 152. Maximum Product Subarray](https://leetcode.com/problems/maximum-product-subarray/description/)

def maxProduct(self, nums):  
 """  
 :type nums: List[int]  
 :rtype: int  
 """  
 if not nums:  
 return 0  
 max\_here = nums[0]  
 min\_here = nums[0]  
 max\_so\_far = nums[0]  
 for num in nums[1:]:  
 max\_here \*= num  
 min\_here \*= num  
 max\_here, min\_here = max(max\_here, min\_here, num), min(max\_here, min\_here, num)  
 max\_so\_far = max(max\_here, max\_so\_far)  
 return max\_so\_far

### 2.15. 判断一个数是否为2的幂次

[Leetcode 231. Power of Two](https://leetcode.com/problems/power-of-two/#/description)

def isPowerOfTwo(n):  
 """  
 :type n: int  
 :rtype: bool  
 """  
 return n > 0 and 0x80000000 % n == 0

### 2.16. 判断一个数是否为4的幂次

[Leetcode 342. Power of Four](https://leetcode.com/problems/power-of-four/#/description)

def isPowerOfFour(num):  
 """  
 :type num: int  
 :rtype: bool  
 """  
 return num > 0 and num & (num - 1) == 0 and num & 0x55555555 == num

### 2.17. 最近公共祖先问题

#### 2.17.1. 二叉树的最近公共祖先

[Leetcode 236. Lowest Common Ancestor of a Binary Tree](https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-tree/description/)

public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {  
 if (root == null || root == p || root == q) {  
 return root;  
 }  
 TreeNode left = lowestCommonAncestor(root.left, p, q);  
 TreeNode right = lowestCommonAncestor(root.right, p, q);  
 if (left != null && right != null) {  
 return root;  
 }  
 return left == null ? right : left;  
}

#### 2.17.2. 二叉搜索树的最近公共祖先

[Leetcode 235. Lowest Common Ancestor of a Binary Search Tree](https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-search-tree/description/)

public TreeNode lowestCommonAncestor(TreeNode root, TreeNode p, TreeNode q) {  
 while (root != null) {  
 if ((p.val < root.val && q.val > root.val)  
 || (p.val > root.val && q.val < root.val)  
 || p.val == root.val || q.val == root.val) {  
 return root;  
 }  
 if (p.val < root.val && q.val < root.val) {  
 root = root.left;  
 }  
 if (p.val > root.val && q.val > root.val) {  
 root = root.right;  
 }  
 }  
 return null;  
}

### 2.18. 挑选数组中不连续的元素，使其和最大

[Leetcode 198. House Robber](https://leetcode.com/problems/house-robber/#/description)

public int rob(int[] nums) {  
 if (nums.length == 0) {  
 return 0;  
 }  
 int[] results = new int[nums.length + 1];  
 results[0] = 0;  
 results[1] = nums[0];  
 for (int i = 2; i <= nums.length; i++) {  
 results[i] = Math.max(nums[i - 1] + results[i - 2], results[i - 1]);  
 }  
 return results[nums.length];  
}

### 2.19. 镜像二叉树

[Leetcode 101. Symmetric Tree](https://leetcode.com/problems/symmetric-tree/#/description)

def isSymmetric(root):  
 """  
 :type root: TreeNode  
 :rtype: bool  
 """  
  
 def symmetric(left, right):  
 if not left and not right:  
 return True  
 if left and right and left.val == right.val:  
 return symmetric(left.left, right.right) and symmetric(left.right, right.left)  
 return False  
  
 if not root:  
 return True  
 return symmetric(root.left, root.right)

### 2.20. 判断二叉树是否平衡

[Leetcode 110. Balanced Binary Tree](https://leetcode.com/problems/balanced-binary-tree/#/description)

def isBalanced(root):  
 """  
 :type root: TreeNode  
 :rtype: bool  
 """  
  
 def balanced(root):  
 if not root:  
 return 0  
 left = balanced(root.left)  
 right = balanced(root.right)  
 if left == -1 or right == -1 or abs(left - right) > 1:  
 return -1  
 return 1 + max(left, right)  
  
 return balanced(root) != -1

### 2.21. 二分法求平方根

[Leetcode 367. Valid Perfect Square](https://leetcode.com/problems/valid-perfect-square/#/description)

public int mySqrt(int x) {  
 if (x == 0) {  
 return 0;  
 }  
 int low = 1;  
 int high = x;  
 while (low <= high) {  
 int mid = (low + high) / 2;  
 if (mid <= x / mid) {  
 low = mid + 1;  
 } else {  
 high = mid - 1;  
 }  
 }  
 return low - 1;  
}

### 2.22. n!求末尾0的个数

[Leetcode 172. Factorial Trailing Zeroes](https://leetcode.com/problems/factorial-trailing-zeroes/#/description)

public int trailingZeroes(int n) {  
 int result = 0;  
 long i = 5;  
 while (i <= n) {  
 result += n / i;  
 i \*= 5;  
 }  
 return result;  
}

### 2.23. 有序数组去重

[Leetcode 26. Remove Duplicates from Sorted Array](https://leetcode.com/problems/remove-duplicates-from-sorted-array/#/description)

def removeDuplicates(self, nums):  
 """  
 :type nums: List[int]  
 :rtype: int  
 """  
 if not nums:  
 return 0  
 i = 0  
 for j in range(1, len(nums)):  
 if nums[j] != nums[i]:  
 i += 1  
 nums[i] = nums[j]  
 return i + 1

### 2.24. 链表判断环

[Leetcode 141. Linked List Cycle](https://leetcode.com/problems/linked-list-cycle/#/description)

def hasCycle(head):  
 """  
 :type head: ListNode  
 :rtype: bool  
 """  
 if head is None or head.next is None:  
 return False  
 walker = head  
 runner = head.next  
 while runner.next and runner.next.next:  
 walker = walker.next  
 runner = runner.next.next  
 if walker == runner:  
 return True  
 return False

### 2.25. 有环链表获取环起始结点

[Leetcode 142. Linked List Cycle II](https://leetcode.com/problems/linked-list-cycle-ii/description/)

def detectCycle(head):  
 """  
 :type head: ListNode  
 :rtype: ListNode  
 """  
 if head is None or head.next is None:  
 return None  
 walker = head  
 runner = head  
 while runner and runner.next:  
 walker = walker.next  
 runner = runner.next.next  
 if walker == runner:  
 break  
 if runner is None:  
 return None  
 walker = head  
 while walker != runner:  
 walker = walker.next  
 runner = runner.next  
 return walker

### 2.26. 判断是否为回文数

[Leetcode 9. Palindrome Number](https://leetcode.com/problems/palindrome-number/#/description)

public boolean isPalindrome(int x) {  
 if (x == 0) {  
 return true;  
 }  
 if (x < 0 || x % 10 == 0) {  
 return false;  
 }  
 int palindrome = 0;  
 while (palindrome < x) {  
 palindrome = palindrome \* 10 + x % 10;  
 x /= 10;  
 }  
 return x == palindrome || x == palindrome / 10;  
}

### 2.27. 是否存在二叉树根到叶子节点路径和为指定值

[Leetcode 112. Path Sum](https://leetcode.com/problems/path-sum/#/description)

def hasPathSum(root, sum):  
 """  
 :type root: TreeNode  
 :type sum: int  
 :rtype: bool  
 """  
 if root is None:  
 return False  
 if root.val == sum and root.left is None and root.right is None:  
 return True  
  
 return hasPathSum(root.left, sum - root.val) or hasPathSum(root.right, sum - root.val)

### 2.28. 找出字符串中所有变位词

[Leetcode 438. Find All Anagrams in a String](https://leetcode.com/problems/find-all-anagrams-in-a-string/#/description)

public List<Integer> findAnagrams(String s, String p) {  
 List<Integer> list = new ArrayList<>();  
 if (s == null || s.length() == 0 || p == null || p.length() == 0) return list;  
 int[] hash = new int[256];  
 for (char c : p.toCharArray()) {  
 hash[c]++;  
 }  
 int left = 0, right = 0, count = p.length();  
 while (right < s.length()) {  
 if (hash[s.charAt(right)] >= 1) {  
 count--;  
 }  
 hash[s.charAt(right)]--;  
 right++;  
 if (count == 0) {  
 list.add(left);  
 }  
 if (right - left == p.length()) {  
 if (hash[s.charAt(left)] >= 0) {  
 count++;  
 }  
 hash[s.charAt(left)]++;  
 left++;  
 }  
 }  
 return list;  
}

### 2.29. 判断两个字符串是否为同构字符串

[Leetcode 205. Isomorphic Strings](https://leetcode.com/problems/isomorphic-strings/#/description)

public boolean isIsomorphic(String s, String t) {  
 int[] m1 = new int[256];  
 int[] m2 = new int[256];  
 for (int i = 0; i < s.length(); i++) {  
 char k1 = s.charAt(i);  
 char k2 = t.charAt(i);  
 if (m1[k1] != m2[k2]) {  
 return false;  
 }  
 m1[k1] = i + 1;  
 m2[k2] = i + 1;  
 }  
 return true;  
}

### 2.30. 判断一个数是否为其全部约数的和

[Leetcode 507. Perfect Number](https://leetcode.com/problems/perfect-number/#/description)

def checkPerfectNumber(num):  
 """  
 :type num: int  
 :rtype: bool  
 """  
 if num == 1:  
 return False  
 result = 0  
 i = 2  
 while i \* i <= num:  
 if num % i == 0:  
 result += i  
 result += num // i  
 i += 1  
 result += 1  
 return result == num

### 2.31. 打乱数组元素

[Leetcode 384. Shuffle an Array](https://leetcode.com/problems/shuffle-an-array/#/description)

public void shuffle(int[] nums) {  
 int length = nums.length;  
 Random random = new Random();  
 for (int i = 0; i < length; i++) {  
 int swapIndex = random.nextInt(length);  
 int temp = nums[i];  
 nums[i] = nums[swapIndex];  
 nums[swapIndex] = temp;  
 }  
 return nums;  
}

### 2.32. 生成不重复随机序列

Random random = new Random();  
  
public int[] randomSequence(int n) {  
 int[] sequence = new int[n];  
 int[] result = new int[n];  
 for (int i = 0; i < n; i++) {  
 sequence[i] = i;  
 }  
 int last = n - 1;  
 for (int i = 0; i < n; i++) {  
 int index = random.nextInt(last + 1);  
 result[i] = sequence[index];  
 sequence[index] = sequence[last];  
 last--;  
 }  
 return result;  
}

### 2.33. 使用随机生成0到5的函数，实现随机生成0到7的函数

Random random = new Random();  
  
int random5(){  
 return random.nextInt(6);  
}  
int random7(){  
 while(true){  
 int temp1=random5();  
 int temp2=random5();  
 int n=temp1\*6+temp2;  
 if(n<8){  
 return n;  
 }  
 }  
}

### 2.34. 计算数组A、B、C、D中有多少种(i, j, k, l)使得A[i] + B[j] + C[k] + D[l] = 0

[Leetcode 454. 4Sum II](https://leetcode.com/problems/4sum-ii/#/description)

def fourSumCount(A, B, C, D):  
 """  
 :type A: List[int]  
 :type B: List[int]  
 :type C: List[int]  
 :type D: List[int]  
 :rtype: int  
 """  
 sumABDict = {}  
 for a in A:  
 for b in B:  
 sumAB = a + b  
 if sumAB not in sumABDict:  
 sumABDict[sumAB] = 0  
 sumABDict[sumAB] += 1  
 result = 0  
 for c in C:  
 for d in D:  
 sumCD = -c - d  
 if sumCD in sumABDict:  
 result += sumABDict[sumCD]  
 return result

### 2.35. 计算给出的多个时间中，最小的时间差

[Leetcode 539. Minimum Time Difference](https://leetcode.com/problems/minimum-time-difference/#/description)

public int findMinDifference(List<String> timePoints) {  
 boolean[] mark = new boolean[24 \* 60];  
 for (String time : timePoints) {  
 String[] t = time.split(":");  
 int h = Integer.parseInt(t[0]);  
 int m = Integer.parseInt(t[1]);  
 if (mark[h \* 60 + m]) return 0;  
 mark[h \* 60 + m] = true;  
 }  
 int prev = 0, min = Integer.MAX\_VALUE;  
 int first = Integer.MAX\_VALUE, last = Integer.MIN\_VALUE;  
 for (int i = 0; i < 24 \* 60; i++) {  
 if (mark[i]) {  
 if (first != Integer.MAX\_VALUE) {  
 min = Math.min(min, i - prev);  
 }  
 first = Math.min(first, i);  
 last = Math.max(last, i);  
 prev = i;  
 }  
 }  
 min = Math.min(min, (24 \* 60 - last + first));  
 return min;  
}

### 2.36. 计算[0, 10^n)中各位数字不同的数的个数

[Leetcode 357. Count Numbers with Unique Digits](https://leetcode.com/problems/count-numbers-with-unique-digits/#/description)

def countNumbersWithUniqueDigits(n):  
 """  
 :type n: int  
 :rtype: int  
 """  
 if n == 0:  
 return 1  
 map = [None] \* (n + 1)  
 map[1] = 10  
 for i in range(2, n + 1):  
 count = 9  
 for j in range(9 - i + 2, 10):  
 count \*= j  
 map[i] = map[i - 1] + count  
 return map[n]

### 2.37. 数组中仅有两个数仅出现一次，其余的数均出现两次，找出这两个数

[Leetcode 260. Single Number III](https://leetcode.com/problems/single-number-iii/#/description)

def singleNumber(nums):  
 """  
 :type nums: List[int]  
 :rtype: List[int]  
 """  
 mixed\_xor = 0  
 for num in nums:  
 mixed\_xor ^= num  
 diff = mixed\_xor & (mixed\_xor - 1) ^ mixed\_xor  
 first = 0  
 second = 0  
 for num in nums:  
 if diff & num:  
 first ^= num  
 else:  
 second ^= num  
 return [first, second]

### 2.38. 找出递增数列中第n位的数字

[Leetcode 400. Nth Digit](https://leetcode.com/problems/nth-digit/#/description)

def findNthDigit(n):  
 """  
 :type n: int  
 :rtype: int  
 """  
 digits = 1  
 start = 1  
 length = 9  
 while (n > digits \* length):  
 n -= digits \* length  
 digits += 1  
 start \*= 10  
 length \*= 10  
 n -= 1  
 number = start + n // digits  
 return int(str(number)[n % digits])

### 2.39. 计算小于n的质数的个数

[Leetcode 204. Count Primes](https://leetcode.com/problems/count-primes/#/description)

def countPrimes(n):  
 """  
 :type n: int  
 :rtype: int  
 """  
 if n < 2:  
 return 0  
 primes = [True] \* n  
 primes[0] = False  
 primes[1] = False  
 for i in range(2, int(n \*\* 0.5) + 1):  
 if primes[i]:  
 for j in range(i \* i, n, i):  
 primes[j] = False  
 return sum(primes)

### 2.40. 由部分（可重复）给定数组元素相加得到指定值的情况数

[Leetcode 377. Combination Sum IV](https://leetcode.com/problems/combination-sum-iv/#/description)

public int combinationSum4(int[] nums, int target) {  
 int[] results = new int[target + 1];  
 results[0] = 1;  
 for (int i = 1; i <= target; i++) {  
 for (int num : nums) {  
 if (i - num >= 0) {  
 results[i] += results[i - num];  
 }  
 }  
 }  
 return results[target];  
}

### 2.41. 在有序矩阵中寻找第k大的数

[Leetcode 378. Kth Smallest Element in a Sorted Matrix](https://leetcode.com/problems/kth-smallest-element-in-a-sorted-matrix/#/description)

public int kthSmallest(int[][] matrix, int k) {  
 class Item {  
 int value;  
 int i;  
 int j;  
  
 public Item(int value, int i, int j) {  
 this.value = value;  
 this.i = i;  
 this.j = j;  
 }  
 }  
 int m = matrix.length;  
 int n = matrix[0].length;  
 PriorityQueue<Item> minHeap = new PriorityQueue<>(new Comparator<Item>() {  
 @Override  
 public int compare(Item o1, Item o2) {  
 return o1.value - o2.value;  
 }  
 });  
 for (int j = 0; j < n; j++) {  
 minHeap.add(new Item(matrix[0][j], 0, j));  
 }  
 while (!minHeap.isEmpty()) {  
 Item item = minHeap.poll();  
 if (item.i < m - 1) {  
 minHeap.add(new Item(matrix[item.i + 1][item.j], item.i + 1, item.j));  
 }  
 k--;  
 if (k == 0) {  
 return item.value;  
 }  
 }  
 return 0;  
}

### 2.42. 求n对括号的所有排列情况

[Leetcode 22. Generate Parentheses](https://leetcode.com/problems/generate-parentheses/#/description)

def generateParenthesis(n):  
 """  
 :type n: int  
 :rtype: List[str]  
 """  
  
 def parenthesis(result, current, left, right):  
 if left == 0 and right == 0:  
 result.append(current)  
 elif left >= 0 and right >= left:  
 parenthesis(result, current + '(', left - 1, right)  
 parenthesis(result, current + ')', left, right - 1)  
  
 result = []  
 parenthesis(result, '', n, n)  
 return result

### 2.43. 给定字符串数组，求最大不相交的两个字符串的长度积

[Leetcode 318. Maximum Product of Word Lengths](https://leetcode.com/problems/maximum-product-of-word-lengths/#/description)

public int maxProduct(String[] words) {  
 int[] bits = new int[words.length];  
 int result = 0;  
 for (int i = 0; i < words.length; i++) {  
 int bit = 0;  
 for (char letter : words[i].toCharArray()) {  
 bit |= 1 << (letter - 'a');  
 }  
 bits[i] = bit;  
 }  
 for (int i = 0; i < bits.length; i++) {  
 for (int j = i + 1; j < bits.length; j++) {  
 if ((bits[i] & bits[j]) == 0 && words[i].length() \* words[j].length() > result) {  
 result = words[i].length() \* words[j].length();  
 }  
 }  
 }  
 return result;  
}

### 2.44. 在由[1, n]组成的长为n + 1的数组中，寻找重复的那个数

[Leetcode 287. Find the Duplicate Number](https://leetcode.com/problems/find-the-duplicate-number/#/description)

public int findDuplicate(int[] nums) {  
 int low = 1;  
 int high = nums.length - 1;  
 while (low < high) {  
 int mid = (low + high) / 2;  
 int count = 0;  
 for (int num : nums) {  
 if (num <= mid) {  
 count++;  
 }  
 }  
 if (count <= mid) {  
 low = mid + 1;  
 } else {  
 high = mid;  
 }  
 }  
 return low;  
}

### 2.45. 求给定数组的所有排列情况

[Leetcode 46. Permutations](https://leetcode.com/problems/permutations/#/description)

public List<List<Integer>> permute(int[] nums) {  
 List<List<Integer>> result = new ArrayList<>();  
 result.add(new ArrayList<>());  
 for (int i = 0; i < nums.length; i++) {  
 List<List<Integer>> newResult = new ArrayList<>();  
 for (int j = 0; j <= i; j++) {  
 for (List<Integer> item : result) {  
 List<Integer> newItem = new ArrayList<>(item);  
 newItem.add(j, nums[i]);  
 newResult.add(newItem);  
 }  
 }  
 result = newResult;  
 }  
 return result;  
}

### 2.46. 求给定（元素可重复）数组的所有排列情况

[Leetcode 47. Permutations II](https://leetcode.com/problems/permutations-ii/description/)

def permuteUnique(self, nums):  
 """  
 :type nums: List[int]  
 :rtype: List[List[int]]  
 """  
 result = []  
 result.append([])  
 for i, num in enumerate(nums):  
 current\_result = []  
 for item in result:  
 for j in range(i + 1):  
 if j > 0 and num == item[j - 1]:  
 break  
 temp = item[:]  
 temp.insert(j, num)  
 current\_result.append(temp)  
 result = current\_result  
 return result

### 2.47. 最长回文子序列

[Leetcode 516. Longest Palindromic Subsequence](https://leetcode.com/problems/longest-palindromic-subsequence/#/description)

public int longestPalindromeSubseq(String s) {  
 int[][] dp = new int[s.length()][s.length()];  
 for (int i = s.length() - 1; i >= 0; i--) {  
 dp[i][i] = 1;  
 for (int j = i + 1; j < s.length(); j++) {  
 if (s.charAt(i) == s.charAt(j)) {  
 dp[i][j] = dp[i + 1][j - 1] + 2;  
 } else {  
 dp[i][j] = Math.max(dp[i + 1][j], dp[i][j - 1]);  
 }  
 }  
 }  
 return dp[0][s.length() - 1];  
}

### 2.48. 将[1, n]的数按字典排序

[Leetcode 386. Lexicographical Numbers](https://leetcode.com/problems/lexicographical-numbers/#/description)

def lexicalOrder(n):  
 """  
 :type n: int  
 :rtype: List[int]  
 """  
  
 def dfs(result, current, n):  
 if current > n:  
 return  
 result.append(current)  
 current \*= 10  
 if current <= n:  
 for i in range(10):  
 if current + i <= n:  
 dfs(result, current + i, n)  
  
 result = []  
 for i in range(1, 10):  
 dfs(result, i, n)  
 return result

### 2.49. 二叉搜索树迭代器

[Leetcode 173. Binary Search Tree Iterator](https://leetcode.com/problems/binary-search-tree-iterator/#/description)

class BSTIterator(object):  
 def \_\_init\_\_(self, root):  
 """  
 :type root: TreeNode  
 """  
 self.stack = []  
 while root:  
 self.stack.append(root)  
 root = root.left  
  
 def hasNext(self):  
 """  
 :rtype: bool  
 """  
 return self.stack  
  
 def next(self):  
 """  
 :rtype: int  
 """  
 node = self.stack.pop()  
 result = node.val  
 node = node.right  
 while node:  
 self.stack.append(node)  
 node = node.left  
 return result

### 2.50. 由[1, n]组成的二叉搜索树的个数

[Leetcode 96. Unique Binary Search Trees](https://leetcode.com/problems/unique-binary-search-trees/#/description)

public int numTrees(int n) {  
 int[] results = new int[n + 1];  
 results[0] = 1;  
 for (int i = 1; i <= n; i++) {  
 for (int j = 0; j < i; j++) {  
 results[i] += results[j] \* results[i - j - 1];  
 }  
 }  
 return results[n];  
}

### 2.51. 格雷码

[Leetcode 89. Gray Code](https://leetcode.com/problems/gray-code/#/description)

def grayCode(n):  
 """  
 :type n: int  
 :rtype: List[int]  
 """  
 return [i ^ (i >> 1) for i in range(1 << n)]

### 2.52. 从m x n棋盘左上角走到右下角的路径总数

[Leetcode 62. Unique Paths](https://leetcode.com/problems/unique-paths/#/description)

public int uniquePaths(int m, int n) {  
 int[][] dp = new int[m][n];  
 for (int i = 0; i < m; i++) {  
 dp[i][0] = 1;  
 }  
 for (int j = 0; j < n; j++) {  
 dp[0][j] = 1;  
 }  
 for (int i = 1; i < m; i++) {  
 for (int j = 1; j < n; j++) {  
 dp[i][j] = dp[i - 1][j] + dp[i][j - 1];  
 }  
 }  
 return dp[m - 1][n - 1];  
}

### 2.53. 求集合的所有子集

[Leetcode 78. Subsets](https://leetcode.com/problems/subsets/#/description)

def subsets(nums):  
 """  
 :type nums: List[int]  
 :rtype: List[List[int]]  
 """  
 result = []  
 for i in range(1 << len(nums)):  
 current = []  
 for j in range(len(nums)):  
 if i & 1 << j:  
 current.append(nums[j])  
 result.append(current)  
 return result

### 2.54. 求（元素可重复）集合的所有子集

[Leetcode 90. Subsets II](https://leetcode.com/problems/subsets-ii/description/)

def subsetsWithDup(self, nums):  
 """  
 :type nums: List[int]  
 :rtype: List[List[int]]  
 """  
 result = [[]]  
 nums.sort()  
 last\_start = 0  
 for i, num in enumerate(nums):  
 start = last\_start if (i > 0 and nums[i] == nums[i - 1]) else 0  
 last\_start = len(result)  
 for item in result[start:last\_start]:  
 temp = item[:]  
 temp.append(num)  
 result.append(temp)  
 return result

### 2.55. 判断数组arr中是否存在arr[i] < arr[j] < arr[k]

[Leetcode 334. Increasing Triplet Subsequence](https://leetcode.com/problems/increasing-triplet-subsequence/#/description)

public boolean increasingTriplet(int[] nums) {  
 int first = Integer.MAX\_VALUE;  
 int second = Integer.MAX\_VALUE;  
 for (int num : nums) {  
 if (num < first) {  
 first = num;  
 } else if (num < second && num > first) {  
 second = num;  
 } else if (num > second) {  
 return true;  
 }  
 }  
 return false;  
}

### 2.56. 查找数组中第k大的数

[Leetcode 215. Kth Largest Element in an Array](https://leetcode.com/problems/kth-largest-element-in-an-array/#/description)

def find\_kth\_largest(nums, k):  
 def partition(array, low, high):  
 pivot = array[low]  
 while low < high:  
 while low < high and array[high] >= pivot:  
 high -= 1  
 array[low] = array[high]  
 while low < high and array[low] <= pivot:  
 low += 1  
 array[high] = array[low]  
 array[low] = pivot  
 return low  
  
 random.shuffle(nums)  
 k = len(nums) - k  
 low = 0  
 high = len(nums) - 1  
 while low < high:  
 index = partition(nums, low, high)  
 if index < k:  
 low = index + 1  
 elif index > k:  
 high = index - 1  
 else:  
 break  
 return nums[k]

### 2.57. 链表删除倒数第n个结点

[Leetcode 19. Remove Nth Node From End of List](https://leetcode.com/problems/remove-nth-node-from-end-of-list/#/description)

def removeNthFromEnd(head, n):  
 """  
 :type head: ListNode  
 :type n: int  
 :rtype: ListNode  
 """  
 first = head  
 second = head  
 prev = head  
 while n:  
 first = first.next  
 n -= 1  
 while first:  
 first = first.next  
 prev = second  
 second = second.next  
 if second == head:  
 return head.next  
 prev.next = prev.next.next  
 return head

### 2.58. 判断数组是否可以拼接为正方形

[Leetcode 473. Matchsticks to Square](https://leetcode.com/problems/matchsticks-to-square/#/description)

public boolean makesquare(int[] nums) {  
 class Utils {  
 boolean square(int[] nums, int[] edges, int start) {  
 if (start == nums.length) {  
 return edges[0] == 0 && edges[1] == 0 && edges[2] == 0 && edges[3] == 0;  
 }  
 int current = nums[start];  
 for (int i = 0; i < edges.length; i++) {  
 if (edges[i] >= current) {  
 edges[i] -= current;  
 if (square(nums, edges, start + 1)) {  
 return true;  
 }  
 edges[i] += current;  
 }  
 }  
 return false;  
 }  
 }  
 if (nums.length < 4) {  
 return false;  
 }  
 int sumNums = 0;  
 for (int num : nums) {  
 sumNums += num;  
 }  
 if (sumNums % 4 != 0) {  
 return false;  
 }  
 int[] edges = new int[4];  
 Arrays.fill(edges, sumNums / 4);  
 Arrays.sort(nums);  
 for (int left = 0, right = nums.length - 1; left < right; left++, right--) {  
 int temp = nums[left];  
 nums[left] = nums[right];  
 nums[right] = temp;  
 }  
 Utils utils = new Utils();  
 return utils.square(nums, edges, 0);  
}

### 2.59. 快速pow()

[Leetcode 372. Super Pow](https://leetcode.com/problems/super-pow/#/description)

public int normalPow(int a, int b) {  
 int result = 1;  
 while (b != 0) {  
 if (b % 2 != 0)  
 result = result \* a % M;  
 a = a \* a % M;  
 b /= 2;  
 }  
 return result;  
}  
  
public int superPow(int a, int[] b) {  
 a %= M;  
 int result = 1;  
 for (int i = b.length - 1; i >= 0; i--) {  
 result = result \* normalPow(a, b[i]) % M;  
 a = normalPow(a, 10);  
 }  
 return result;

### 2.60. 判断数组是否可以分为相等的两部分

[Leetcode 416. Partition Equal Subset Sum](https://leetcode.com/problems/partition-equal-subset-sum/#/description)

public boolean canPartition(int[] nums) {  
 int target = 0;  
 for (int num : nums) {  
 target += num;  
 }  
 if (target % 2 == 1) {  
 return false;  
 }  
 target /= 2;  
 boolean[][] dp = new boolean[nums.length + 1][target + 1];  
 dp[0][0] = true;  
 for (int i = 1; i <= nums.length; i++) {  
 for (int j = 0; j <= target; j++) {  
 dp[i][j] = dp[i - 1][j];  
 if (j >= nums[i - 1]) {  
 dp[i][j] = dp[i][j] || dp[i - 1][j - nums[i - 1]];  
 }  
 }  
 }  
 return dp[nums.length][target];  
}

### 2.61. 在矩阵中查找指定值

[Leetcode 240. Search a 2D Matrix II](https://leetcode.com/problems/search-a-2d-matrix-ii/#/description)

def searchMatrix(matrix, target):  
 """  
 :type matrix: List[List[int]]  
 :type target: int  
 :rtype: bool  
 """  
 if not matrix:  
 return False  
 i = 0  
 j = len(matrix[0]) - 1  
 while i < len(matrix) and j >= 0:  
 current = matrix[i][j]  
 if current == target:  
 return True  
 elif current > target:  
 j -= 1  
 else:  
 i += 1  
 return False

### 2.62. 将矩阵顺时针旋转90度

[Leetcode 48. Rotate Image](https://leetcode.com/problems/rotate-image/#/description)

def rotate(matrix):  
 """  
 :type matrix: List[List[int]]  
 :rtype: void Do not return anything, modify matrix in-place instead.  
 """  
 m = len(matrix)  
 n = len(matrix[0])  
 for i in range(m // 2):  
 matrix[i], matrix[m - i - 1] = matrix[m - i - 1], matrix[i]  
 for i in range(m):  
 for j in range(i + 1, n):  
 matrix[i][j], matrix[j][i] = matrix[j][i], matrix[i][j]

### 2.63. 最长递增子序列

[Leetcode 300. Longest Increasing Subsequence](https://leetcode.com/problems/longest-increasing-subsequence/#/description)

def lengthOfLIS(nums):  
 """  
 :type nums: List[int]  
 :rtype: int  
 """  
 if not nums:  
 return 0  
 dp = [1] \* len(nums)  
 for i in range(1, len(nums)):  
 for j in range(i):  
 if nums[i] > nums[j]:  
 dp[i] = max(dp[i], dp[j] + 1)  
  
 return max(dp)

### 2.64. 求二维矩阵由左上角到右下角的最小和

[Leetcode 64. Minimum Path Sum](https://leetcode.com/problems/minimum-path-sum/#/description)

public int minPathSum(int[][] grid) {  
 int m = grid.length;  
 int n = grid[0].length;  
 int[][] dp = new int[m][n];  
 dp[0][0] = grid[0][0];  
 for (int i = 1; i < m; i++) {  
 dp[i][0] = dp[i - 1][0] + grid[i][0];  
 }  
 for (int j = 1; j < n; j++) {  
 dp[0][j] = dp[0][j - 1] + grid[0][j];  
 }  
 for (int i = 1; i < m; i++) {  
 for (int j = 1; j < n; j++) {  
 dp[i][j] = Math.min(dp[i - 1][j], dp[i][j - 1]) + grid[i][j];  
 }  
 }  
 return dp[m - 1][n - 1];  
}

### 2.65. 将由0、1、2组成的数组排序

[Leetcode 75. Sort Colors](https://leetcode.com/problems/sort-colors/#/description)

def sortColors(self, nums):  
 """  
 :type nums: List[int]  
 :rtype: void Do not return anything, modify nums in-place instead.  
 """  
 left = 0  
 right = len(nums) - 1  
 middle = 0  
 while middle <= right:  
 while nums[middle] == 2 and middle < right:  
 nums[middle], nums[right] = nums[right], nums[middle]  
 right -= 1  
 while nums[middle] == 0 and middle > left:  
 nums[middle], nums[left] = nums[left], nums[middle]  
 left += 1  
 middle += 1

### 2.66. 求第n位仅由指定因数组成的数字

[Leetcode 313. Super Ugly Number](https://leetcode.com/problems/super-ugly-number/#/description)

public int nthSuperUglyNumber(int n, int[] primes) {  
 int[] factors = new int[primes.length];  
 Arrays.fill(factors, 1);  
 int[] indexes = new int[primes.length];  
 Arrays.fill(indexes, -1);  
 int[] ugly = new int[n];  
 for (int i = 0; i < n; i++) {  
 int theMin = Integer.MAX\_VALUE;  
 for (int factor : factors) {  
 theMin = Math.min(theMin, factor);  
 }  
 ugly[i] = theMin;  
 for (int j = 0; j < factors.length; j++) {  
 if (ugly[i] == factors[j]) {  
 indexes[j]++;  
 factors[j] = ugly[indexes[j]] \* primes[j];  
 }  
 }  
 }  
 return ugly[ugly.length - 1];  
}

### 2.67. 数组中查找极大值

[Leetcode 162. Find Peak Elementr](https://leetcode.com/problems/find-peak-element/description/)

public int findPeakElement(int[] nums) {  
 if (nums.length == 1 || nums[0] > nums[1]) {  
 return 0;  
 }  
 if (nums[nums.length - 2] < nums[nums.length - 1]) {  
 return nums.length - 1;  
 }  
 int low = 0;  
 int high = nums.length - 1;  
 while (low < high) {  
 int mid = (low + high) / 2;  
 if (nums[mid] < nums[mid + 1]) {  
 low = mid + 1;  
 } else {  
 high = mid;  
 }  
 }  
 return low;  
}

### 2.68. 判断n能否由平方数的和组成

[Leetcode 279. Perfect Squares](https://leetcode.com/problems/perfect-squares/#/description)

def numSquares(self, n):  
 """  
 :type n: int  
 :rtype: int  
 """  
 dp = [n] \* (n + 1)  
 dp[0] = 0  
 for i in range(1, n + 1):  
 j = 1  
 while j \* j <= i:  
 dp[i] = min(dp[i], dp[i - j \* j] + 1)  
 j += 1  
 return dp[n]

### 2.69. 删除二叉搜索树中的结点

[Leetcode 450. Delete Node in a BST](https://leetcode.com/problems/delete-node-in-a-bst/#/description)

def deleteNode(root, key):  
 """  
 :type root: TreeNode  
 :type key: int  
 :rtype: TreeNode  
 """  
 if not root:  
 return None  
 if root.val > key:  
 root.left = deleteNode(root.left, key)  
 elif root.val < key:  
 root.right = deleteNode(root.right, key)  
 else:  
 if not root.left:  
 return root.right  
 elif not root.right:  
 return root.left  
 else:  
 replace = root.right  
 while replace.left:  
 replace = replace.left  
 root.val = replace.val  
 root.right = deleteNode(root.right, root.val)  
 return root

### 2.70. 检查字符串是否是合法的二叉树前序遍历结果

[Leetcode 331. Verify Preorder Serialization of a Binary Tree](https://leetcode.com/problems/verify-preorder-serialization-of-a-binary-tree/#/description)

def isValidSerialization(self, preorder):  
 """  
 :type preorder: str  
 :rtype: bool  
 """  
 degree = -1  
 for node in preorder.split(','):  
 degree += 1  
 if degree > 0:  
 return False  
 if node != '#':  
 degree -= 2  
 return degree == 0

### 2.71. 中缀表达式转后缀表达式

String infixToPostfix(String infix) {  
 Map<Character, Integer> priority = new HashMap<>();  
 priority.put('(', 1);  
 priority.put(')', -1);  
 priority.put('+', 2);  
 priority.put('-', 2);  
 priority.put('\*', 3);  
 priority.put('/', 3);  
 priority.put('^', 3);  
 StringBuilder builder = new StringBuilder();  
 Stack<Character> stack = new Stack<>();  
 for (char c : infix.toCharArray()) {  
 if (priority.containsKey(c)) {  
 if (c == '(') {  
 stack.push(c);  
 } else if (c == ')') {  
 while (!stack.empty() && stack.peek() != '(') {  
 builder.append(stack.pop());  
 }  
 if (!stack.empty()) {  
 stack.pop();  
 }  
 } else {  
 while (!stack.empty() && priority.get(stack.peek()) >= priority.get(c)) {  
 builder.append(stack.pop());  
 }  
 stack.push(c);  
 }  
 } else {  
 builder.append(c);  
 }  
 }  
 while (!stack.empty()) {  
 builder.append(stack.pop());  
 }  
 return builder.toString();  
}

### 2.72. 排列的下一种情况

[Leetcode 31. Next Permutation](https://leetcode.com/problems/next-permutation/description/)

def nextPermutation(self, nums):  
 """  
 :type nums: List[int]  
 :rtype: void Do not return anything, modify nums in-place instead.  
 """  
 i = len(nums) - 2  
 while i >= 0 and nums[i] >= nums[i + 1]:  
 i -= 1  
 if i < 0:  
 nums.sort()  
 return  
 j = len(nums) - 1  
 while nums[j] <= nums[i]:  
 j -= 1  
 nums[i], nums[j] = nums[j], nums[i]  
 nums[i + 1:] = sorted(nums[i + 1:])

### 2.73. 求旋转有序数组的最小值

[Leetcode 153. Find Minimum in Rotated Sorted Array](https://leetcode.com/problems/find-minimum-in-rotated-sorted-array/description/)

def findMin(self, nums):  
 """  
 :type nums: List[int]  
 :rtype: int  
 """  
 length = len(nums)  
 low = 0  
 high = length - 1  
 while low < high:  
 mid = (low + high) // 2  
 if nums[mid] > nums[high]:  
 low = mid + 1  
 else:  
 high = mid  
 return nums[low]

### 2.74. 在旋转有序数组中查找指定值

[Leetcode 33. Search in Rotated Sorted Array](https://leetcode.com/problems/search-in-rotated-sorted-array/description/)

def search(self, nums, target):  
 """  
 :type nums: List[int]  
 :type target: int  
 :rtype: int  
 """  
 length = len(nums)  
 low = 0  
 high = length - 1  
 while low < high:  
 mid = (low + high) // 2  
 if nums[mid] > nums[high]:  
 low = mid + 1  
 else:  
 high = mid  
 rotate = low  
 low = 0  
 high = length - 1  
 while low <= high:  
 mid = (low + high) // 2  
 real\_mid = (mid + rotate) % length  
 if nums[real\_mid] == target:  
 return real\_mid  
 if nums[real\_mid] < target:  
 low = mid + 1  
 else:  
 high = mid - 1  
 return -1

### 2.75. 在旋转（元素可重复）有序数组中查找指定值

[Leetcode 81. Search in Rotated Sorted Array II](https://leetcode.com/problems/search-in-rotated-sorted-array-ii/description/)

def search(self, nums, target):  
 """  
 :type nums: List[int]  
 :type target: int  
 :rtype: bool  
 """  
 if not nums:  
 return False  
 low = 0  
 high = len(nums) - 1  
 while low < high:  
 mid = (low + high) // 2  
 if nums[mid] == target:  
 return True  
 if nums[mid] > nums[high]:  
 if nums[low] <= target < nums[mid]:  
 high = mid  
 else:  
 low = mid + 1  
 elif nums[mid] < nums[high]:  
 if nums[mid] < target <= nums[high]:  
 low = mid + 1  
 else:  
 high = mid  
 else:  
 high -= 1  
 return nums[low] == target

### 2.76. 有序数组中查找特定元素的第一个和最后一个索引

[Leetcode 34. Search for a Range](https://leetcode.com/problems/search-for-a-range/description/)

def searchRange(self, nums, target):  
 """  
 :type nums: List[int]  
 :type target: int  
 :rtype: List[int]  
 """  
  
 def search(nums, target):  
 low = 0  
 high = len(nums) - 1  
 while low <= high:  
 mid = (low + high) // 2  
 if nums[mid] < target:  
 low = mid + 1  
 else:  
 high = mid - 1  
 return low  
  
 if not nums:  
 return [-1, -1]  
 low = search(nums, target)  
 if low == len(nums) or nums[low] != target:  
 return [-1, -1]  
 high = search(nums, target + 1)  
 high -= 1  
 return [low, high]

### 2.77. 有序数组中查找指定值的插入位置

[Leetcode 35. Search Insert Position](https://leetcode.com/problems/search-insert-position/description/)

def searchInsert(self, nums, target):  
 """  
 :type nums: List[int]  
 :type target: int  
 :rtype: int  
 """  
 low = 0  
 high = len(nums) - 1  
 while low < high:  
 mid = (low + high) // 2  
 if nums[mid] == target:  
 return mid  
 if nums[mid] > target:  
 high = mid - 1  
 else:  
 low = mid + 1  
 return low

### 2.78. 实现pow(x, n)

[Leetcode 50. Pow(x, n)](https://leetcode.com/problems/powx-n/description/)

def myPow(self, x, n):  
 """  
 :type x: float  
 :type n: int  
 :rtype: float  
 """  
 if not n:  
 return 1  
 if n < 0:  
 n = -n  
 x = 1 / x  
 if n % 2:  
 return x \* myPow(x \* x, n // 2)  
 return myPow(x \* x, n // 2)

### 2.79. 第k个排列

[Leetcode 60. Permutation Sequence](https://leetcode.com/problems/permutation-sequence/description/)

def getPermutation(self, n, k):  
 """  
 :type n: int  
 :type k: int  
 :rtype: str  
 """  
 factor = [0] \* n  
 factor[0] = 1  
 for i in range(1, n):  
 factor[i] = factor[i - 1] \* i  
 result = ''  
 k -= 1  
 numbers = [i for i in range(1, 10)]  
 for i in range(n - 1, -1, -1):  
 number = k // factor[i]  
 k %= factor[i]  
 result += str(numbers[number])  
 del numbers[number]  
 return result

### 2.80. 链表向右循环移动k个位置

[Leetcode 61. Rotate List](https://leetcode.com/problems/rotate-list/description/)

def rotateRight(self, head, k):  
 """  
 :type head: ListNode  
 :type k: int  
 :rtype: ListNode  
 """  
 if not head:  
 return head  
 tail = head  
 length = 1  
 while tail.next:  
 tail = tail.next  
 length += 1  
 if k % length:  
 tail.next = head  
 step = length - k%length - 1  
 new\_head = head  
 for i in range(step):  
 new\_head = new\_head.next  
 head = new\_head.next  
 new\_head.next = None  
 return head

### 2.81. 先序遍历和中序遍历构建二叉树

[Leetcode 105. Construct Binary Tree from Preorder and Inorder Traversal](https://leetcode.com/problems/construct-binary-tree-from-preorder-and-inorder-traversal/description/)

def buildTree(self, preorder, inorder):  
 """  
 :type preorder: List[int]  
 :type inorder: List[int]  
 :rtype: TreeNode  
 """  
 if not preorder or not inorder:  
 return None  
 node = TreeNode(preorder.pop(0))  
 index = inorder.index(node.val)  
 node.left = buildTree(preorder, inorder[:index])  
 node.right = buildTree(preorder, inorder[index + 1:])  
 return node

### 2.82. 中序遍历和后序遍历构建二叉树

[Leetcode 106. Construct Binary Tree from Inorder and Postorder Traversal](https://leetcode.com/problems/construct-binary-tree-from-inorder-and-postorder-traversal/description/)

def buildTree(self, inorder, postorder):  
 """  
 :type inorder: List[int]  
 :type postorder: List[int]  
 :rtype: TreeNode  
 """  
 if not inorder or not postorder:  
 return None  
 node = TreeNode(postorder.pop())  
 index\_inorder = inorder.index(node.val)  
 node.right = buildTree(inorder[index\_inorder + 1:], postorder)  
 node.left = buildTree(inorder[:index\_inorder], postorder)  
 return node

### 2.83. 求杨辉三角第k行的值

[Leetcode 119. Pascal's Triangle II](https://leetcode.com/problems/pascals-triangle-ii/description/)

def getRow(self, rowIndex):  
 """  
 :type rowIndex: int  
 :rtype: List[int]  
 """  
 result = [0] \* (rowIndex + 1)  
 result[0] = 1  
 for row in range(1, rowIndex + 1):  
 for i in range(row, 0, -1):  
 result[i] += result[i - 1]  
 return result

### 2.84. 克隆图

[Leetcode 133. Clone Graph](https://leetcode.com/problems/clone-graph/description/)

def cloneGraph(self, node):  
 def clone(node, memory):  
 if node.label not in memory:  
 copy = UndirectedGraphNode(node.label)  
 memory[copy.label] = copy  
 for neighbor in node.neighbors:  
 copy.neighbors.append(clone(neighbor, memory))  
 return memory[node.label]  
 if not node:  
 return node  
 memory = {}  
 return clone(node, memory)

### 2.85. 查找数组中的极大值元素

[Leetcode 162. Find Peak Element](https://leetcode.com/problems/find-peak-element/description/)

def findPeakElement(self, nums):  
 """  
 :type nums: List[int]  
 :rtype: int  
 """  
 if len(nums) == 1 or nums[0] > nums[1]:  
 return 0  
 if nums[-2] < nums[-1]:  
 return len(nums) - 1  
 low = 0  
 high = len(nums) - 1  
 while low < high:  
 mid = (low + high) // 2  
 if nums[mid] < nums[mid + 1]:  
 low = mid + 1  
 else:  
 high = mid  
 return low

### 2.86. 将数组合并拼接为最大的数

[Leetcode 179. Largest Number](https://leetcode.com/problems/largest-number/description/)

def largestNumber(self, nums):  
 strs = [str(num) for num in nums]  
 strs.sort(lambda x, y: 1 if x + y < y + x else -1 if x + y > y + x else 0)  
 if strs[0] == '0':  
 return '0'  
 return ''.join(strs)

### 2.87. 计算完全二叉树的结点数

[Leetcode 222. Count Complete Tree Nodes](https://leetcode.com/problems/count-complete-tree-nodes/description/)

def countNodes(self, root):  
 """  
 :type root: TreeNode  
 :rtype: int  
 """  
 if not root:  
 return 0  
 depth\_left = 0  
 node\_left = root  
 while node\_left:  
 depth\_left += 1  
 node\_left = node\_left.left  
 depth\_right = 0  
 node\_right = root  
 while node\_right:  
 depth\_right += 1  
 node\_right = node\_right.right  
 if depth\_left == depth\_right:  
 return 2 \*\* depth\_left - 1  
 return 1 + self.countNodes(root.left) + self.countNodes(root.right)

### 2.88. 加减乘除（无括号）计算器

[Leetcode 227. Basic Calculator II](https://leetcode.com/problems/basic-calculator-ii/description/)

def calculate(self, s):  
 """  
 :type s: str  
 :rtype: int  
 """  
 result = 0  
 pre\_number = 0  
 pre\_operator = '+'  
 i = 0  
 while i < len(s):  
 while i < len(s) and s[i] == ' ':  
 i += 1  
 current = 0  
 while i < len(s) and s[i] >= '0' and s[i] <= '9':  
 current = current \* 10 + ord(s[i]) - ord('0')  
 i += 1  
 if pre\_operator == '+':  
 result += pre\_number  
 pre\_number = current  
 elif pre\_operator == '-':  
 result += pre\_number  
 pre\_number = -current  
 elif pre\_operator == '\*':  
 pre\_number \*= current  
 else:  
 temp = pre\_number  
 pre\_number /= current  
 if temp < 0 and temp % current:  
 pre\_number += 1  
 while i < len(s) and s[i] == ' ':  
 i += 1  
 if i < len(s):  
 pre\_operator = s[i]  
 i += 1  
 result += pre\_number  
 return result

### 2.89. 加减（带括号）计算器

[Leetcode 224. Basic Calculator](https://leetcode.com/problems/basic-calculator/description/)

def calculate(self, s):  
 """  
 :type s: str  
 :rtype: int  
 """  
 stack = []  
 result = 0  
 current = 0  
 sign = 1  
 for letter in s:  
 if letter.isdigit():  
 current = current \* 10 + ord(letter) - ord('0')  
 elif letter in ['+', '-']:  
 result += sign \* current  
 current = 0  
 sign = 1 if letter == '+' else -1  
 elif letter == '(':  
 stack.append(result)  
 stack.append(sign)  
 sign = 1  
 result = 0  
 elif letter == ')':  
 result += sign \* current  
 result \*= stack.pop()  
 result += stack.pop()  
 current = 0  
 result += sign \* current  
 return result

### 2.90. 二叉搜索树中第k小的数

[Leetcode 230. Kth Smallest Element in a BST](https://leetcode.com/problems/kth-smallest-element-in-a-bst/description/)

def kthSmallest(self, root, k):  
 """  
 :type root: TreeNode  
 :type k: int  
 :rtype: int  
 """  
 self.k = k  
 self.result = None  
  
 def smallest(root):  
 if not root:  
 return  
 smallest(root.left)  
 self.k -= 1  
 if not self.k:  
 self.result = root.val  
 return  
 smallest(root.right)  
  
 smallest(root)  
 return self.result

### 2.91. 求（元素可更新）数组指定区间的和

[Leetcode 307. Range Sum Query - Mutable](https://leetcode.com/problems/range-sum-query-mutable/description/)

class NumArray(object):  
 class Node:  
 def \_\_init\_\_(self, start, end):  
 self.start = start  
 self.end = end  
 self.left = None  
 self.right = None  
 self.sum = 0  
  
 def build(self, nums, start, end):  
 if start > end:  
 return None  
 node = self.Node(start, end)  
 if start == end:  
 node.sum = nums[start]  
 else:  
 mid = (start + end) // 2  
 node.left = self.build(nums, start, mid)  
 node.right = self.build(nums, mid + 1, end)  
 node.sum = node.left.sum + node.right.sum  
 return node  
  
 def \_\_init\_\_(self, nums):  
 """  
 :type nums: List[int]  
 """  
 self.root = self.build(nums, 0, len(nums) - 1)  
  
 def update(self, i, val):  
 """  
 :type i: int  
 :type val: int  
 :rtype: void  
 """  
  
 def upd(node, i, val):  
 if node.start == node.end:  
 node.sum = val  
 else:  
 mid = (node.start + node.end) // 2  
 if i <= mid:  
 upd(node.left, i, val)  
 else:  
 upd(node.right, i, val)  
 node.sum = node.left.sum + node.right.sum  
  
 upd(self.root, i, val)  
  
 def sumRange(self, i, j):  
 """  
 :type i: int  
 :type j: int  
 :rtype: int  
 """  
  
 def sum\_ran(node, start, end):  
 if node.start == start and node.end == end:  
 return node.sum  
 mid = (node.start + node.end) // 2  
 if end <= mid:  
 return sum\_ran(node.left, start, end)  
 if start > mid:  
 return sum\_ran(node.right, start, end)  
 return sum\_ran(node.left, start, mid) + sum\_ran(node.right, mid + 1, end)  
  
 return sum\_ran(self.root, i, j)

### 2.92. 求数组中任意两个元素异或的最大值

[Leetcode 421. Maximum XOR of Two Numbers in an Array](https://leetcode.com/problems/maximum-xor-of-two-numbers-in-an-array/description/)

def findMaximumXOR(self, nums):  
 """  
 :type nums: List[int]  
 :rtype: int  
 """  
 result = 0  
 mask = 0  
 for i in range(31, -1, -1):  
 mask |= 1 << i  
 nums\_set = set()  
 for num in nums:  
 nums\_set.add(num & mask)  
 current = result | (1 << i)  
 for num in nums\_set:  
 if num ^ current in nums\_set:  
 result = current  
 break  
 return result

### 2.93. 二叉搜索树的序列化和反序列化

[Leetcode 449. Serialize and Deserialize BST](https://leetcode.com/problems/serialize-and-deserialize-bst/description/)

class Codec:  
  
 def serialize(self, root):  
 """Encodes a tree to a single string.  
  
 :type root: TreeNode  
 :rtype: str  
 """  
 result = ''  
 stack = []  
 while root or stack:  
 if root:  
 result += str(root.val) + ' '  
 stack.append(root)  
 root = root.left  
 else:  
 node = stack.pop()  
 root = node.right  
 return result  
  
 def deserialize(self, data):  
 """Decodes your encoded data to tree.  
  
 :type data: str  
 :rtype: TreeNode  
 """  
 data = [int(item) for item in data.split()]  
  
 def construct(data, i, j):  
 if i == j:  
 return None  
 node = TreeNode(data[i])  
 split\_index = i + 1  
 while split\_index < j and data[split\_index] < data[i]:  
 split\_index += 1  
 node.left = construct(data, i + 1, split\_index)  
 node.right = construct(data, split\_index, j)  
 return node  
  
 return construct(data, 0, len(data))

### 2.94. 轮流取数判断先取者是否必胜

[Leetcode 486. Predict the Winner](https://leetcode.com/problems/predict-the-winner/description/)

def PredictTheWinner(self, nums):  
 """  
 :type nums: List[int]  
 :rtype: bool  
 """  
  
 def predict(nums, start, end, first, second):  
 if start > end:  
 return first >= second  
 return not predict(nums, start + 1, end, second, first + nums[start]) \  
 or not predict(nums, start, end - 1, second, first + nums[end])  
  
 if len(nums) <= 1:  
 return True  
 return predict(nums, 0, len(nums) - 1, 0, 0)

### 2.95. 一串数加上分别加上加号或减号使其结果等于指定值

[Leetcode 494. Target Sum](https://leetcode.com/problems/target-sum/description/)

def findTargetSumWays(self, nums, S):  
 """  
 :type nums: List[int]  
 :type S: int  
 :rtype: int  
 """  
 sum\_nums = sum(nums)  
 if S > sum\_nums or S < -sum\_nums:  
 return 0  
 map = [0] \* (2 \* sum\_nums + 1)  
 map[sum\_nums] = 1  
 for num in nums:  
 new\_map = [0] \* (2 \* sum\_nums + 1)  
 for i in range(2 \* sum\_nums + 1):  
 if map[i]:  
 new\_map[i + num] += map[i]  
 new\_map[i - num] += map[i]  
 map = new\_map  
 return map[sum\_nums + S]

### 2.96. 有序数组中除一个数仅出现一次外，其他数均出现两次，求这个数

[Leetcode 540. Single Element in a Sorted Array](https://leetcode.com/problems/single-element-in-a-sorted-array/description/)

def singleNonDuplicate(self, nums):  
 """  
 :type nums: List[int]  
 :rtype: int  
 """  
 low = 0  
 high = len(nums) - 1  
 while low < high:  
 mid = (low + high) // 2  
 if (mid % 2 == 0 and nums[mid] == nums[mid + 1]) or (mid % 2 != 0 and nums[mid] == nums[mid - 1]):  
 low = mid + 1  
 else:  
 high = mid - 1  
 return nums[low]

### 2.97. 最长任意元素差最大为1的子序列

[Leetcode 594. Longest Harmonious Subsequence](https://leetcode.com/problems/longest-harmonious-subsequence/description/)

def findLHS(self, nums):  
 """  
 :type nums: List[int]  
 :rtype: int  
 """  
 memory = {}  
 for num in nums:  
 if num not in memory:  
 memory[num] = 0  
 memory[num] += 1  
 result = 0  
 for key, value in memory.items():  
 if key + 1 in memory:  
 result = max(result, value + memory[key + 1])  
 return result

### 2.98. 数组中寻找任意三个元素的最大积

[Leetcode 628. Maximum Product of Three Numbers](https://leetcode.com/problems/maximum-product-of-three-numbers/description/)

def maximumProduct(self, nums):  
 """  
 :type nums: List[int]  
 :rtype: int  
 """  
 max\_1 = -sys.maxsize  
 max\_2 = -sys.maxsize  
 max\_3 = -sys.maxsize  
 min\_1 = sys.maxsize  
 min\_2 = sys.maxsize  
 for num in nums:  
 if num > max\_1:  
 max\_3 = max\_2  
 max\_2 = max\_1  
 max\_1 = num  
 elif num > max\_2:  
 max\_3 = max\_2  
 max\_2 = num  
 elif num > max\_3:  
 max\_3 = num  
 if num < min\_1:  
 min\_2 = min\_1  
 min\_1 = num  
 elif num < min\_2:  
 min\_2 = num  
 return max(max\_1 \* max\_2 \* max\_3, max\_1 \* min\_1 \* min\_2)

### 2.99. 生成结点个数为n的所有二叉搜索树

[Leetcode 95. Unique Binary Search Trees II](https://leetcode.com/problems/unique-binary-search-trees-ii/description/)

def generateTrees(self, n):  
 """  
 :type n: int  
 :rtype: List[TreeNode]  
 """  
 def generate(start, end):  
 result = []  
 if start > end:  
 return [None]  
 for i in range(start, end + 1):  
 lefts = generate(start, i - 1)  
 rights = generate(i + 1, end)  
 for left in lefts:  
 for right in rights:  
 root = TreeNode(i)  
 root.left = left  
 root.right = right  
 result.append(root)  
 return result  
  
 if n == 0:  
 return []  
 return generate(1, n)

### 2.100. 数组中除一个数仅出现一次外，其他数均出现三次，求这个数

[Leetcode 137. Single Number II](https://leetcode.com/problems/single-number-ii/description/)

def singleNumber(self, nums):  
 """  
 :type nums: List[int]  
 :rtype: int  
 """  
 x1 = 0  
 x2 = 0  
 for num in nums:  
 x2 ^= x1 & num  
 x1 ^= num  
 mask = ~(x1 & x2)  
 x2 &= mask  
 x1 &= mask  
 return x1

### 2.101. 将一个数拆分为多个数的和，并使这些数的积最大

[Leetcode 343. Integer Break](https://leetcode.com/problems/integer-break/description/)

def integerBreak(self, n):  
 """  
 :type n: int  
 :rtype: int  
 """  
 if n == 2:  
 return 1  
 if n == 3:  
 return 2  
 if n == 4:  
 return 4  
 result = 1  
 while n > 4:  
 n -= 3  
 result \*= 3  
 if n == 4:  
 result \*= 4  
 elif n == 3:  
 result \*= 3  
 elif n == 2:  
 result \*= 2  
 return result

### 2.102. 轮流取数先取者是否必胜

[Leetcode 464. Can I Win](https://leetcode.com/problems/can-i-win/description/)

def canIWin(self, maxChoosableInteger, desiredTotal):  
 """  
 :type maxChoosableInteger: int  
 :type desiredTotal: int  
 :rtype: bool  
 """  
 def to\_key(choosed):  
 result = 1  
 for item in choosed:  
 result <<= 1  
 if item:  
 result |= 1  
 return result  
  
 def win(choosed, memory, total):  
 key = to\_key(choosed)  
 if key in memory:  
 return memory[key]  
 for i in range(1, len(choosed)):  
 if not choosed[i]:  
 choosed[i] = True  
 if total <= i or not win(choosed, memory, total - i):  
 memory[key] = True  
 choosed[i] = False  
 return True  
 choosed[i] = False  
 memory[key] = False  
 return False  
  
 if desiredTotal == 0:  
 return True  
 if (1 + maxChoosableInteger) \* maxChoosableInteger // 2 < desiredTotal:  
 return False  
 return win([False] \* (maxChoosableInteger + 1), {}, desiredTotal)

### 2.103. 区间调度问题

#### 2.103.1. 最多区间调度

[Leetcode 633. Sum of Square Numbers](https://leetcode.com/problems/sum-of-square-numbers/description/)

def maxNonoverlapIntervals(self, intervals):  
 """  
 :type points: List[List[int]]  
 :rtype: int  
 """  
 intervals.sort(key=lambda interval: interval[1])  
 current\_start = -sys.maxsize  
 result = 0  
 for interval in intervals:  
 if interval[0] > current\_start:  
 current\_start = interval[1]  
 result += 1  
 return result

### 2.104. 寻找数组中是否存在nums[i]和nums[j]差的绝对值最大为t，i和j差的绝对值最大为k

[Leetcode 220. Contains Duplicate III](https://leetcode.com/problems/contains-duplicate-iii/description/)

def containsNearbyAlmostDuplicate(self, nums, k, t):  
 """  
 :type nums: List[int]  
 :type k: int  
 :type t: int  
 :rtype: bool  
 """  
 if k < 1 or t < 0:  
 return False  
 buckets = {}  
 for i, num in enumerate(nums):  
 adjusted\_num = num + 0x80000000  
 bucket = adjusted\_num // (t + 1)  
 if bucket in buckets \  
 or (bucket - 1 in buckets and abs(adjusted\_num - buckets[bucket - 1]) <= t) \  
 or (bucket + 1 in buckets and abs(adjusted\_num - buckets[bucket + 1]) <= t):  
 return True  
 buckets[bucket] = adjusted\_num  
 if i >= k:  
 del buckets[(nums[i - k] + 0x80000000) // (t + 1)]  
 return False

### 2.105. 猜数游戏，猜错需要付钱，求确保猜中的最小代价

[Leetcode 375. Guess Number Higher or Lower II](https://leetcode.com/problems/guess-number-higher-or-lower-ii/description/)

def getMoneyAmount(self, n):  
 """  
 :type n: int  
 :rtype: int  
 """  
 def get\_money(start, end, memory):  
 if start >= end:  
 return 0  
 if memory[start][end] != 0:  
 return memory[start][end]  
 result = sys.maxsize  
 for i in range(start, end + 1):  
 current = i + max(get\_money(start, i - 1, memory), get\_money(i + 1, end, memory))  
 result = min(result, current)  
 memory[start][end] = result  
 return result  
  
 memory = [[0] \* (n + 1) for \_ in range(n + 1)]  
 return get\_money(0, n, memory)

### 2.106. 寻找数组中是否存在i < j < k使得ai < ak < aj

[Leetcode 456. 132 Pattern](https://leetcode.com/problems/132-pattern/description/)

def find132pattern(self, nums):  
 """  
 :type nums: List[int]  
 :rtype: bool  
 """  
 if len(nums) < 3:  
 return False  
 stack = []  
 for num in nums:  
 if not stack or stack[-1][0] > num:  
 stack.append((num, num))  
 elif stack[-1][0] < num:  
 if stack[-1][1] > num:  
 return True  
 the\_min = stack[-1][0]  
 stack.pop()  
 while stack and stack[-1][1] <= num:  
 stack.pop()  
 if stack and stack[-1][0] < num:  
 return True  
 stack.append((the\_min, num))  
 return False

### 2.107. 矩阵上从某一点经过最多N次移动出界外的路径总数

[Leetcode 576. Out of Boundary Paths](https://leetcode.com/problems/out-of-boundary-paths/description/)

def findPaths(self, m, n, N, i, j):  
 """  
 :type m: int  
 :type n: int  
 :type N: int  
 :type i: int  
 :type j: int  
 :rtype: int  
 """  
 dp = [[[0] \* (n) for \_ in range(m)] for \_ in range(N + 1)]  
 moves = ((-1, 0), (1, 0), (0, -1), (0, 1))  
 for k in range(1, N + 1):  
 for a in range(m):  
 for b in range(n):  
 for move in moves:  
 row = a + move[0]  
 col = b + move[1]  
 if row == -1 or col == -1 or row == m or col == n:  
 dp[k][a][b] += 1  
 else:  
 dp[k][a][b] += dp[k - 1][row][col]  
 dp[k][a][b] %= 1000000007  
 return dp[N][i][j]

### 2.108. 每次操作删除一个字符，求使得两个字符串相同的最少操作次数

[Leetcode 583. Delete Operation for Two Strings](https://leetcode.com/problems/delete-operation-for-two-strings/description/)

def minDistance(self, word1, word2):  
 """  
 :type word1: str  
 :type word2: str  
 :rtype: int  
 """  
 l1 = len(word1)  
 l2 = len(word2)  
 dp = [[0] \* (l2 + 1) for \_ in range(l1 + 1)]  
 for i in range(1, l1 + 1):  
 for j in range(1, l2 + 1):  
 if word1[i - 1] == word2[j - 1]:  
 dp[i][j] = dp[i - 1][j - 1] + 1  
 else:  
 dp[i][j] = max(dp[i][j - 1], dp[i - 1][j])  
 max\_length = dp[l1][l2]  
 return l1 - max\_length + l2 - max\_length

### 2.109. 求字符串的（不同下标）回文字串的数目

[Leetcode 647. Palindromic Substrings](https://leetcode.com/problems/palindromic-substrings/description/)

def countSubstrings(self, s):  
 """  
 :type s: str  
 :rtype: int  
 """  
 self.result = 0  
  
 def count(s, left, right):  
 while left >= 0 and right < len(s) and s[left] == s[right]:  
 self.result += 1  
 left -= 1  
 right += 1  
  
 for i in range(len(s)):  
 count(s, i, i)  
 count(s, i, i + 1)  
 return self.result

### 2.110. 打印二叉树

[Leetcode 655. Print Binary Tree](https://leetcode.com/problems/print-binary-tree/description/)

def printTree(self, root):  
 """  
 :type root: TreeNode  
 :rtype: List[List[str]]  
 """  
 def get\_depth(root):  
 if not root:  
 return 0  
 return max(get\_depth(root.left), get\_depth(root.right)) + 1  
  
 depth = get\_depth(root)  
 self.result = [[''] \* (2 \*\* depth - 1) for \_ in range(depth)]  
  
 def print\_tree(root, row, depth, col):  
 self.result[row][col] = str(root.val)  
 row += 1  
 if root.left:  
 print\_tree(root.left, row, depth, col - 2 \*\* (depth - row - 1))  
 if root.right:  
 print\_tree(root.right, row, depth, col + 2 \*\* (depth - row - 1))  
  
 print\_tree(root, 0, depth, 2 \*\* (depth - 1) - 1)  
 return self.result

### 2.111. 是否可以仅改变一个元素，使得数组为单调递增数组

[Leetcode 665. Non-decreasing Array](https://leetcode.com/problems/non-decreasing-array/description/)

def checkPossibility(self, nums):  
 """  
 :type nums: List[int]  
 :rtype: bool  
 """  
 is\_modified = False  
 for i in range(1, len(nums)):  
 if nums[i - 1] > nums[i]:  
 if is\_modified:  
 return False  
 is\_modified = True  
 if i < 2 or nums[i - 2] <= nums[i]:  
 nums[i - 1] = nums[i]  
 else:  
 nums[i] = nums[i - 1]  
 return True

### 2.112. 仅调换一次一个数的两个数字，求最大值

[Leetcode 670. Maximum Swap](https://leetcode.com/problems/maximum-swap/description/)

def maximumSwap(self, num):  
 """  
 :type num: int  
 :rtype: int  
 """  
 num\_str = list(str(num))  
 last\_index = {}  
 for i, item in enumerate(num\_str):  
 last\_index[item] = i  
 for i, item in enumerate(num\_str):  
 for digit in '987654321':  
 if digit > item and digit in last\_index and last\_index[digit] > i:  
 num\_str[i], num\_str[last\_index[digit]] = num\_str[last\_index[digit]], num\_str[i]  
 return int(''.join(num\_str))  
 return num

### 2.113. 检查带有通配符的括号序列是否合法

[Leetcode 678. Valid Parenthesis String](https://leetcode.com/problems/valid-parenthesis-string/description/)

def checkValidString(self, s):  
 """  
 :type s: str  
 :rtype: bool  
 """  
 min\_close = 0  
 max\_close = 0  
 for letter in s:  
 if letter == '(':  
 min\_close += 1  
 max\_close += 1  
 elif letter == ')':  
 if min\_close > 0:  
 min\_close -= 1  
 max\_close -= 1  
 if max\_close < 0:  
 return False  
 else:  
 if min\_close > 0:  
 min\_close -= 1  
 max\_close += 1  
 return min\_close == 0

### 2.114. 两个有序数组的中位数

[Leetcode 4. Median of Two Sorted Arrays](https://leetcode.com/problems/median-of-two-sorted-arrays/description/)

def findMedianSortedArrays(self, nums1, nums2):  
 """  
 :type nums1: List[int]  
 :type nums2: List[int]  
 :rtype: float  
 """  
 if len(nums1) > len(nums2):  
 return self.findMedianSortedArrays(nums2, nums1)  
 short\_length = len(nums1)  
 long\_length = len(nums2)  
 short\_left = 0  
 short\_right = len(nums1)  
 while short\_left <= short\_right:  
 short\_mid = (short\_left + short\_right) // 2  
 long\_mid = (short\_length + long\_length + 1) // 2 - short\_mid  
 if short\_mid < short\_length and nums1[short\_mid] < nums2[long\_mid - 1]:  
 short\_left = short\_mid + 1  
 elif short\_mid > 0 and nums1[short\_mid - 1] > nums2[long\_mid]:  
 short\_right = short\_mid - 1  
 else:  
 max\_left = None  
 if short\_mid == 0:  
 max\_left = nums2[long\_mid - 1]  
 elif long\_mid == 0:  
 max\_left = nums1[short\_mid - 1]  
 else:  
 max\_left = max(nums1[short\_mid - 1], nums2[long\_mid - 1])  
 if (short\_length + long\_length) % 2:  
 return max\_left  
 min\_right = None  
 if short\_mid == short\_length:  
 min\_right = nums2[long\_mid]  
 elif long\_mid == long\_length:  
 min\_right = nums1[short\_mid]  
 else:  
 min\_right = min(nums1[short\_mid], nums2[long\_mid])  
 return (max\_left + min\_right) / 2.0

### 2.115. 带有.和\*的正则表达式匹配

[Leetcode 10. Regular Expression Matching](https://leetcode.com/problems/regular-expression-matching/description/)

def isMatch(self, s, p):  
 """  
 :type s: str  
 :type p: str  
 :rtype: bool  
 """  
 length\_s = len(s)  
 length\_p = len(p)  
 dp = [[False] \* (length\_p + 1) for i in range(length\_s + 1)]  
 dp[0][0] = True  
 for i in range(1, length\_s + 1):  
 dp[i][0] = False  
 for j in range(1, length\_p + 1):  
 dp[0][j] = j > 1 and p[j - 1] == '\*' and dp[0][j - 2]  
 for i in range(1, length\_s + 1):  
 for j in range(1, length\_p + 1):  
 if p[j - 1] == '\*':  
 dp[i][j] = dp[i][j - 2] or ((p[j - 2] == s[i - 1] or p[j - 2] == '.') and dp[i - 1][j])  
 else:  
 dp[i][j] = dp[i - 1][j - 1] and (p[j - 1] == s[i - 1] or p[j - 1] == '.')  
 return dp[length\_s][length\_p]

### 2.116. 合并多个有序链表

[Leetcode 23. Merge k Sorted Lists](https://leetcode.com/problems/merge-k-sorted-lists/description/)

# Definition for singly-linked list.  
# class ListNode(object):  
# def \_\_init\_\_(self, x):  
# self.val = x  
# self.next = None  
  
def mergeKLists(self, lists):  
 """  
 :type lists: List[ListNode]  
 :rtype: ListNode  
 """  
 def partition(lists, start, end):  
 if start == end:  
 return lists[start]  
 if start < end:  
 mid = (start + end) // 2  
 list\_1 = partition(lists, start, mid)  
 list\_2 = partition(lists, mid + 1, end)  
 return merge(list\_1, list\_2)  
 return None  
  
 def merge(list\_1, list\_2):  
 fake\_head = ListNode(0)  
 current = fake\_head  
 while list\_1 and list\_2:  
 if list\_1.val < list\_2.val:  
 current.next = list\_1  
 list\_1 = list\_1.next  
 else:  
 current.next = list\_2  
 list\_2 = list\_2.next  
 current = current.next  
 current.next = list\_1 if list\_1 else list\_2  
 return fake\_head.next  
  
 return partition(lists, 0, len(lists) - 1)

### 2.117. 最长合法括号字串

[Leetcode 32. Longest Valid Parentheses](https://leetcode.com/problems/longest-valid-parentheses/description/)

def longestValidParentheses(self, s):  
 """  
 :type s: str  
 :rtype: int  
 """  
 left = 0  
 right = 0  
 result = 0  
 for item in s:  
 if item == '(':  
 left += 1  
 else:  
 right += 1  
 if left == right:  
 result = max(result, left + right)  
 elif left < right:  
 left = 0  
 right = 0  
 left = 0  
 right = 0  
 for item in reversed(s):  
 if item == '(':  
 left += 1  
 else:  
 right += 1  
 if left == right:  
 result = max(result, left + right)  
 elif left > right:  
 left = 0  
 right = 0  
 return result

### 2.118. 带有?和\*的正则表达式匹配

[Leetcode 44. Wildcard Matching](https://leetcode.com/problems/wildcard-matching/description/)

def isMatch(self, s, p):  
 """  
 :type s: str  
 :type p: str  
 :rtype: bool  
 """  
 length\_s = len(s)  
 length\_p = len(p)  
 dp = [[False] \* (length\_p + 1) for i in range(length\_s + 1)]  
 dp[0][0] = True  
 for j in range(1, length\_p + 1):  
 dp[0][j] = dp[0][j - 1] and p[j - 1] == '\*'  
 for i in range(1, length\_s + 1):  
 for j in range(1, length\_p + 1):  
 if p[j - 1] == '\*':  
 dp[i][j] = dp[i - 1][j] or dp[i][j - 1]  
 else:  
 dp[i][j] = dp[i - 1][j - 1] and (p[j - 1] == s[i - 1] or p[j - 1] == '?')  
 return dp[length\_s][length\_p]

### 2.119. 检查字符串是否是合法数字

[Leetcode 65. Valid Number](https://leetcode.com/problems/valid-number/description/)

def isNumber(self, s):  
 """  
 :type s: str  
 :rtype: bool  
 """  
 s = s.strip()  
 has\_dot = False  
 has\_e = False  
 has\_number = False  
 has\_number\_after\_e = False  
 for i, letter in enumerate(s):  
 if letter in '0123456789':  
 has\_number = True  
 if has\_e:  
 has\_number\_after\_e = True  
 elif letter == '.':  
 if has\_dot or has\_e:  
 return False  
 has\_dot = True  
 elif letter == 'e':  
 if has\_e or not has\_number:  
 return False  
 has\_e = True  
 elif letter in '+-':  
 if i != 0 and s[i - 1] != 'e':  
 return False  
 else:  
 return False  
 return (not has\_e ^ has\_number\_after\_e) and has\_number

### 2.120. 通过插入、删除和替换，最少需要多少次操作可以使得两个字符串相同

[Leetcode 72. Edit Distance](https://leetcode.com/problems/edit-distance/description/)

def minDistance(self, word1, word2):  
 """  
 :type word1: str  
 :type word2: str  
 :rtype: int  
 """  
 m = len(word1)  
 n = len(word2)  
 dp = [[0] \* (n + 1) for i in range(m + 1)]  
 for i in range(m + 1):  
 dp[i][0] = i  
 for j in range(n + 1):  
 dp[0][j] = j  
 for i in range(1, m + 1):  
 for j in range(1, n + 1):  
 if word1[i - 1] == word2[j - 1]:  
 dp[i][j] = dp[i - 1][j - 1]  
 else:  
 dp[i][j] = min(dp[i - 1][j - 1], dp[i][j - 1], dp[i - 1][j]) + 1  
 return dp[m][n]

### 2.121. 柱状图中的最大矩形

[Leetcode 84. Largest Rectangle in Histogram](https://leetcode.com/problems/largest-rectangle-in-histogram/description/)

def largestRectangleArea(self, heights):  
 """  
 :type heights: List[int]  
 :rtype: int  
 """  
 stack = []  
 i = 0  
 result = 0  
 while i < len(heights):  
 while stack and heights[i] < heights[stack[-1]]:  
 index = stack.pop()  
 current = heights[index] \* (i - (stack[-1] if stack else -1) - 1)  
 result = max(result, current)  
 stack.append(i)  
 i += 1  
 while stack:  
 index = stack.pop()  
 current = heights[index] \* (i - (stack[-1] if stack else -1) - 1)  
 result = max(result, current)  
 return result

### 2.122. 矩阵中由1组成的最大矩阵

[Leetcode 85. Maximal Rectangle](https://leetcode.com/problems/maximal-rectangle/description/)

def maximalRectangle(self, matrix):  
 """  
 :type matrix: List[List[str]]  
 :rtype: int  
 """  
 if not matrix:  
 return 0  
 heights = [0] \* len(matrix[0])  
 result = 0  
 for row in matrix:  
 for i, item in enumerate(row):  
 if item == '1':  
 heights[i] += 1  
 else:  
 heights[i] = 0  
 stack = []  
 i = 0  
 while i < len(heights):  
 while stack and heights[i] < heights[stack[-1]]:  
 index = stack.pop()  
 current = heights[index] \* (i - (stack[-1] if stack else -1) - 1)  
 result = max(result, current)  
 stack.append(i)  
 i += 1  
 while stack:  
 index = stack.pop()  
 current = heights[index] \* (i - (stack[-1] if stack else -1) - 1)  
 result = max(result, current)  
 return result

### 2.123. 判断字符串是否由给定的两个字符串组合而成

[Leetcode 97. Interleaving String](https://leetcode.com/problems/interleaving-string/description/)

def isInterleave(self, s1, s2, s3):  
 """  
 :type s1: str  
 :type s2: str  
 :type s3: str  
 :rtype: bool  
 """  
 length\_1 = len(s1) + 1  
 length\_2 = len(s2) + 1  
 if len(s1) + len(s2) != len(s3):  
 return False  
 dp = [[False] \* length\_2 for i in range(length\_1)]  
 dp[0][0] = True  
 for i in range(1, length\_1):  
 dp[i][0] = dp[i - 1][0] and s1[i - 1] == s3[i - 1]  
 for j in range(1, length\_2):  
 dp[0][j] = dp[0][j - 1] and s2[j - 1] == s3[j - 1]  
 for i in range(1, length\_1):  
 for j in range(1, length\_2):  
 dp[i][j] = (dp[i - 1][j] and s1[i - 1] == s3[i + j - 1]) or (  
 dp[i][j - 1] and s2[j - 1] == s3[i + j - 1])  
 return dp[length\_1 - 1][length\_2 - 1]

### 2.124. 一个字符串有多少个子序列可以组成另一个字符串

[Leetcode 115. Distinct Subsequences](https://leetcode.com/problems/distinct-subsequences/description/)

def numDistinct(self, s, t):  
 """  
 :type s: str  
 :type t: str  
 :rtype: int  
 """  
 dp = [[None] \* (len(s) + 1) for i in range(len(t) + 1)]  
 dp[0] = [1] \* (len(s) + 1)  
 for i in range(1, len(t) + 1):  
 dp[i][0] = 0  
 for i in range(1, len(t) + 1):  
 for j in range(1, len(s) + 1):  
 dp[i][j] = dp[i][j - 1]  
 if t[i - 1] == s[j - 1]:  
 dp[i][j] += dp[i - 1][j - 1]  
 return dp[len(t)][len(s)]

### 2.125. 二叉树上最大（不需要经过根结点）路径和

[Leetcode 124. Binary Tree Maximum Path Sum](https://leetcode.com/problems/binary-tree-maximum-path-sum/description/)

def maxPathSum(self, root):  
 """  
 :type root: TreeNode  
 :rtype: int  
 """  
 def path\_sum(root, result):  
 if not root:  
 return (0, result)  
 left, result = path\_sum(root.left, result)  
 left = max(0, left)  
 right, result = path\_sum(root.right, result)  
 right = max(0, right)  
 result = max(result, left + right + root.val)  
 return max(left, right) + root.val, result  
  
 return path\_sum(root, -sys.maxsize)[1]

### 2.126. 求将字符串切分为回文字串需要的最少切分次数

[Leetcode 132. Palindrome Partitioning II](https://leetcode.com/problems/palindrome-partitioning-ii/description/)

def minCut(self, s):  
 """  
 :type s: str  
 :rtype: int  
 """  
 length = len(s)  
 cuts = [0] \* length  
 palindrome = [[False] \* length for i in range(length)]  
 for i in range(length):  
 current = i  
 for j in range(i + 1):  
 if s[i] == s[j] and (i - j < 2 or palindrome[j + 1][i - 1]):  
 palindrome[j][i] = True  
 current = 0 if j == 0 else min(current, cuts[j - 1] + 1)  
 cuts[i] = current  
 return cuts[length - 1]

### 2.127. 二维坐标系下一条线上最多多少个点

[Leetcode 149. Max Points on a Line](https://leetcode.com/problems/max-points-on-a-line/description/)

def maxPoints(self, points):  
 """  
 :type points: List[Point]  
 :rtype: int  
 """  
 def transform(a, b):  
 def gcd(a, b):  
 while b:  
 a, b = b, a % b  
 return a  
  
 c = gcd(a, b)  
 a /= c  
 b /= c  
 return (a << 32) | b  
  
 if len(points) < 3:  
 return len(points)  
 result = 0  
 for i, current in enumerate(points):  
 same\_point = 1  
 same\_vertical\_line = 0  
 memory = {}  
 for j in range(i + 1, len(points)):  
 point = points[j]  
 if current.x == point.x and current.y == point.y:  
 same\_point += 1  
 elif current.x == point.x:  
 same\_vertical\_line += 1  
 else:  
 key = transform(point.y - current.y, point.x - current.x)  
 if key not in memory:  
 memory[key] = 0  
 memory[key] += 1  
 result = max(result, same\_vertical\_line + same\_point)  
 if memory:  
 result = max(result, max(memory.values()) + same\_point)  
 return result

### 2.128. 股票交易求最大收益，最多可以进行k次交易

[Leetcode 188. Best Time to Buy and Sell Stock IV](https://leetcode.com/problems/best-time-to-buy-and-sell-stock-iv/description/)

def maxProfit(self, k, prices):  
 """  
 :type k: int  
 :type prices: List[int]  
 :rtype: int  
 """  
 if not prices:  
 return 0  
 if k >= len(prices) // 2:  
 result = 0  
 for i in range(1, len(prices)):  
 if prices[i] > prices[i - 1]:  
 result += prices[i] - prices[i - 1]  
 return result  
 dp = [0] \* len(prices)  
 for time in range(k):  
 current = dp[0] - prices[0]  
 for i in range(1, len(prices)):  
 current = max(current, dp[i] - prices[i])  
 dp[i] = max(dp[i - 1], prices[i] + current)  
 return max(dp)

### 2.129. 在字符串前插入一些字符使得字符串为回文字符串、

[Leetcode 214. Shortest Palindrome](https://leetcode.com/problems/shortest-palindrome/description/)

def shortestPalindrome(self, s):  
 """  
 :type s: str  
 :rtype: str  
 """  
 reverse = s[::-1]  
 for i in range(len(s) + 1):  
 if s.startswith(reverse[i:]):  
 return reverse[:i] + s

### 2.130. 计算从1到n中含有数字1的数的个数

[Leetcode 233. Number of Digit One](https://leetcode.com/problems/number-of-digit-one/description/)

def countDigitOne(self, n):  
 """  
 :type n: int  
 :rtype: int  
 """  
 result = 0  
 times = 1  
 while times <= n:  
 a = n // times  
 b = n % times  
 if a % 10 == 0:  
 result += a // 10 \* times  
 elif a % 10 == 1:  
 result += a // 10 \* times + b + 1  
 else:  
 result += (a // 10 + 1) \* times  
 times \*= 10  
 return result

### 2.131. 移动窗口求每个窗口元素最大值

[Leetcode 239. Sliding Window Maximum](https://leetcode.com/problems/sliding-window-maximum/description/)

def maxSlidingWindow(self, nums, k):  
 """  
 :type nums: List[int]  
 :type k: int  
 :rtype: List[int]  
 """  
 que = collections.deque()  
 result = []  
 for i, num in enumerate(nums):  
 if que and que[0] == i - k:  
 que.popleft()  
 while que and nums[que[-1]] < num:  
 que.pop()  
 que.append(i)  
 if i + 1 >= k:  
 result.append(nums[que[0]])  
 return result

### 2.132. 数字转换为英文单词组合

[Leetcode 273. Integer to English Words](https://leetcode.com/problems/integer-to-english-words/description/)

def numberToWords(self, num):  
 """  
 :type num: int  
 :rtype: str  
 """  
 def convert(num):  
 LESS\_THAN\_20 = ["", "One", "Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", "Ten", "Eleven",  
 "Twelve", "Thirteen", "Fourteen", "Fifteen", "Sixteen", "Seventeen", "Eighteen", "Nineteen"]  
 TENS = ["", "Ten", "Twenty", "Thirty", "Forty", "Fifty", "Sixty", "Seventy", "Eighty", "Ninety"]  
 if num == 0:  
 return ''  
 if num < 20:  
 return LESS\_THAN\_20[num] + ' '  
 if num < 100:  
 return TENS[num // 10] + ' ' + convert(num % 10)  
 return LESS\_THAN\_20[num // 100] + ' Hundred ' + convert(num % 100)  
  
 THOUSANDS = ["", "Thousand", "Million", "Billion"]  
 if num == 0:  
 return 'Zero'  
 i = 0  
 result = ''  
 while num:  
 slice = num % 1000  
 if slice:  
 result = convert(slice) + THOUSANDS[i] + ' ' + result  
 num //= 1000  
 i += 1  
 return result.strip()

### 2.133. 二叉树的序列化和反序列化

[Leetcode 273. Integer to English Words](https://leetcode.com/problems/integer-to-english-words/description/)

class Codec:  
  
 def serialize(self, root):  
 """Encodes a tree to a single string.  
  
 :type root: TreeNode  
 :rtype: str  
 """  
  
 def serial(node, result):  
 if node:  
 result.append(str(node.val))  
 serial(node.left, result)  
 serial(node.right, result)  
 else:  
 result.append('#')  
  
 result = []  
 serial(root, result)  
 return ' '.join(result)  
  
 def deserialize(self, data):  
 """Decodes your encoded data to tree.  
  
 :type data: str  
 :rtype: TreeNode  
 """  
  
 def deserail(que):  
 value = que.popleft()  
 if value == '#':  
 return None  
 node = TreeNode(int(value))  
 node.left = deserail(que)  
 node.right = deserail(que)  
 return node  
  
 que = collections.deque(data.split())  
 return deserail(que)

### 2.134. 移除字符串中的重复字符，并使得最终结果字典序最小

[Leetcode 316. Remove Duplicate Letters](https://leetcode.com/problems/remove-duplicate-letters/description/)

def removeDuplicateLetters(self, s):  
 """  
 :type s: str  
 :rtype: str  
 """  
 stack = []  
 counts = [0] \* 128  
 for letter in s:  
 counts[ord(letter)] += 1  
 exists = [False] \* 128  
 for letter in s:  
 letter\_in\_int = ord(letter)  
 counts[letter\_in\_int] -= 1  
 if exists[letter\_in\_int]:  
 continue  
 while stack and letter\_in\_int < stack[-1] and counts[stack[-1]] > 0:  
 temp = stack.pop()  
 exists[temp] = False  
 stack.append(letter\_in\_int)  
 exists[letter\_in\_int] = True  
 return ''.join(map(chr, stack))

### 2.135. 信封俄罗斯套娃

[Leetcode 354. Russian Doll Envelopes](https://leetcode.com/problems/russian-doll-envelopes/description/)

def maxEnvelopes(self, envelopes):  
 """  
 :type envelopes: List[List[int]]  
 :rtype: int  
 """  
 envelopes.sort(key=lambda x: (x[0], -x[1]))  
 dp = [0] \* len(envelopes)  
 current\_length = 0  
 for envelope in envelopes:  
 low = 0  
 high = current\_length  
 value = envelope[1]  
 while low < high:  
 mid = (low + high) // 2  
 if dp[mid] == value:  
 low = mid  
 break  
 if dp[mid] < value:  
 low = mid + 1  
 else:  
 high = mid  
 dp[low] = value  
 if low == current\_length:  
 current\_length += 1  
 return current\_length

### 2.136. 小矩阵是否能完整拼接为大矩阵

[Leetcode 391. Perfect Rectangle](https://leetcode.com/problems/perfect-rectangle/description/)

def isRectangleCover(self, rectangles):  
 """  
 :type rectangles: List[List[int]]  
 :rtype: bool  
 """  
 x1 = sys.maxsize  
 y1 = sys.maxsize  
 x2 = -sys.maxsize  
 y2 = -sys.maxsize  
 got = set()  
 sum\_up = 0  
 for rectangle in rectangles:  
 a1 = rectangle[0]  
 b1 = rectangle[1]  
 a2 = rectangle[2]  
 b2 = rectangle[3]  
 x1 = min(x1, a1)  
 y1 = min(y1, b1)  
 x2 = max(x2, a2)  
 y2 = max(y2, b2)  
 sum\_up += (a2 - a1) \* (b2 - b1)  
 for i in [0, 2]:  
 for j in [1, 3]:  
 key = str(rectangle[i]) + ' ' + str(rectangle[j])  
 if key in got:  
 got.remove(key)  
 else:  
 got.add(key)  
 return len(got) == 4 \  
 and (str(x1) + ' ' + str(y1)) in got \  
 and (str(x1) + ' ' + str(y2)) in got \  
 and (str(x2) + ' ' + str(y1)) in got \  
 and (str(x2) + ' ' + str(y2)) in got \  
 and sum\_up == (x2 - x1) \* (y2 - y1)

### 2.137. 将数组划分为m份，使得每份和的最大值最小

[Leetcode 410. Split Array Largest Sum](https://leetcode.com/problems/split-array-largest-sum/description/)

def splitArray(self, nums, m):  
 """  
 :type nums: List[int]  
 :type m: int  
 :rtype: int  
 """  
 def cut(nums, mid, m):  
 count = 1  
 current\_sum = 0  
 for num in nums:  
 current\_sum += num  
 if current\_sum > mid:  
 count += 1  
 current\_sum = num  
 if count > m:  
 return False  
 return True  
  
 low = max(nums)  
 high = sum(nums)  
 while low <= high:  
 mid = (low + high) // 2  
 cuts = cut(nums, mid, m)  
 if cuts:  
 high = mid - 1  
 else:  
 low = mid + 1  
 return low

### 2.138. 在1到n中寻找字典序第k小的数

[Leetcode 440. K-th Smallest in Lexicographical Order](https://leetcode.com/problems/k-th-smallest-in-lexicographical-order/description/)

def findKthNumber(self, n, k):  
 """  
 :type n: int  
 :type k: int  
 :rtype: int  
 """  
 current = 1  
 k -= 1  
 while k > 0:  
 steps = 0  
 start = current  
 end = start + 1  
 while start <= n:  
 steps += min(n + 1, end) - start  
 start \*= 10  
 end \*= 10  
 if steps <= k:  
 current += 1  
 k -= steps  
 else:  
 current \*= 10  
 k -= 1  
 return current

### 2.139. 滑动窗口中间值

[Leetcode 480. Sliding Window Median](https://leetcode.com/problems/sliding-window-median/description/)

def medianSlidingWindow(self, nums, k):  
 """  
 :type nums: List[int]  
 :type k: int  
 :rtype: List[float]  
 """  
 min\_heap = []  
 max\_heap = []  
 result = []  
  
 for i, num in enumerate(nums[:k]):  
 heapq.heappush(max\_heap, (-num, i))  
 for \_ in range(k - k // 2):  
 heapq.heappush(min\_heap, (-max\_heap[0][0], max\_heap[0][1]))  
 heapq.heappop(max\_heap)  
 for i in range(k, len(nums)):  
 num = nums[i]  
 result.append(min\_heap[0][0] / 1.0 if k % 2 else (min\_heap[0][0] - max\_heap[0][0]) / 2.0)  
 if num >= min\_heap[0][0]:  
 heapq.heappush(min\_heap, (num, i))  
 if nums[i - k] <= min\_heap[0][0]:  
 heapq.heappush(max\_heap, (-min\_heap[0][0], min\_heap[0][1]))  
 heapq.heappop(min\_heap)  
 else:  
 heapq.heappush(max\_heap, (-num, i))  
 if nums[i - k] >= min\_heap[0][0]:  
 heapq.heappush(min\_heap, (-max\_heap[0][0], max\_heap[0][1]))  
 heapq.heappop(max\_heap)  
 while max\_heap and max\_heap[0][1] <= i - k:  
 heapq.heappop(max\_heap)  
 while min\_heap and min\_heap[0][1] <= i - k:  
 heapq.heappop(min\_heap)  
 result.append(min\_heap[0][0] / 1.0 if k % 2 else (min\_heap[0][0] - max\_heap[0][0]) / 2.0)  
 return result

### 2.140. 多个有序数组，每个数组挑选至少一个数，求能够组成的最小的数组范围

[Leetcode 632. Smallest Range](https://leetcode.com/problems/smallest-range/description/)

def smallestRange(self, nums):  
 """  
 :type nums: List[List[int]]  
 :rtype: List[int]  
 """  
 heap = []  
 end = -sys.maxsize  
 for index, num in enumerate(nums):  
 end = max(end, num[0])  
 heapq.heappush(heap, (num[0], index, 0))  
 start = heap[0][0]  
 temp\_end = end  
 while len(heap) == len(nums):  
 current = heapq.heappop(heap)  
 if current[2] + 1 < len(nums[current[1]]):  
 temp\_end = max(temp\_end, nums[current[1]][current[2] + 1])  
 heapq.heappush(heap, (nums[current[1]][current[2] + 1], current[1], current[2] + 1))  
 if temp\_end - heap[0][0] < end - start:  
 start = heap[0][0]  
 end = temp\_end  
 return [start, end]

### 2.141. 点游戏

[Leetcode 679. 24 Game](https://leetcode.com/problems/24-game/description/)

def judgePoint24(self, nums):  
 """  
 :type nums: List[int]  
 :rtype: bool  
 """  
 EPS = 0.001  
  
 def back\_trace(nums):  
 if len(nums) == 1:  
 return abs(nums[0] - 24) < EPS  
 for i, a in enumerate(nums):  
 for j, b in enumerate(nums[:i]):  
 candidates = [a + b, a - b, b - a, a \* b]  
 if abs(a) > EPS:  
 candidates.append(b / a)  
 if abs(b) > EPS:  
 candidates.append(a / b)  
 del nums[i]  
 del nums[j]  
 for candidate in candidates:  
 nums.append(candidate)  
 if back\_trace(nums):  
 return True  
 nums.pop()  
 nums.insert(j, b)  
 nums.insert(i, a)  
 return False  
  
 return back\_trace(nums)

### 2.142. 数组中寻找三个长为k的不重叠子数组，使得其和最大

[Leetcode 689. Maximum Sum of 3 Non-Overlapping Subarrays](https://leetcode.com/problems/maximum-sum-of-3-non-overlapping-subarrays/description/)

def maxSumOfThreeSubarrays(self, nums, k):  
 """  
 :type nums: List[int]  
 :type k: int  
 :rtype: List[int]  
 """  
 length = len(nums)  
 left\_sums = []  
 temp = 0  
 left\_sums.append(temp)  
 for num in nums:  
 temp += num  
 left\_sums.append(temp)  
 index\_left = [0] \* length  
 current\_max = 0  
 for i in range(k - 1, length):  
 if left\_sums[i + 1] - left\_sums[i + 1 - k] > current\_max:  
 index\_left[i] = i + 1 - k  
 current\_max = left\_sums[i + 1] - left\_sums[i + 1 - k]  
 else:  
 index\_left[i] = index\_left[i - 1]  
 index\_right = [0] \* length  
 current\_max = 0  
 for i in range(length - k, -1, -1):  
 if left\_sums[i + k] - left\_sums[i] >= current\_max:  
 index\_right[i] = i  
 current\_max = left\_sums[i + k] - left\_sums[i]  
 else:  
 index\_right[i] = index\_right[i + 1]  
 the\_max = 0  
 result = [0] \* 3  
 for i in range(k, length - 2 \* k + 1):  
 left = index\_left[i - 1]  
 right = index\_right[i + k]  
 current\_max = left\_sums[left + k] - left\_sums[left] + left\_sums[i + k] - left\_sums[i] + left\_sums[  
 right + k] - left\_sums[right]  
 if current\_max > the\_max:  
 result = [left, i, right]  
 the\_max = current\_max  
 return result

### 2.143. 快速替换10亿条标题中的5万个敏感词

Aho-Corasick算法（AC自动机）：

* 构造一棵Trie树；
* 构造失败指针；
* 模式匹配。

参考：[AC自动机算法详解 - 极限定律 - C++博客](http://www.cppblog.com/mythit/archive/2009/04/21/80633.html)

### 2.144. 轮流取数的必胜策略

问题描述：有2N个自然数，甲乙两人轮流取。一人一次取一个，而且只能取头尾两个数中的一个，取过的数划去，直到2N个数取完，取得的数的总和大的人获胜。那么，先取的人是否有必胜策略？

解决方法：先取的人只要取了奇数位上的数，就把偶数位的数留给后取的人，而且只要坚持取奇数位上的数，就一直会把偶数位上的数留给后取的人。这样就会取走所有奇数位上的数。（这种方法同样可使先取的人把所有偶数位上的数取走）所以，先取的人只要一开始算出是所有偶数位上的数总和大，还是所有奇数位上的数总和大，就一定会胜。

### 2.145. 最大公约数（辗转相除法）

def gcd(a, b):  
 while b:  
 a, b = b, a % b  
 return a

### 2.146. 最小公倍数

def lcm(a, b):  
 return a \* b // gcd(a, b)