刷题笔记 - python - (1-26题)

数组：

数组中重复数字：（python用set的特性就好了）

二维数组的查找：（将矩阵逆时针旋转 45°，并将其转化为图形式，发现其类似于**二叉搜索树**）

```
class Solution:
    def findNumberIn2DArray(self, matrix: List[List[int]], target: int) -> bool:
        i, j = len(matrix) - 1, 0
        while i >= 0 and j < len(matrix[0]):
            if matrix[i][j] > target: i -= 1
            elif matrix[i][j] < target: j += 1
            else: return True
        return False
```

字符串：

替换空格：（python题在这里似乎没有意义，**但C++题必须要掌握**）

链表：

```
class ListNode:
    def __init__(self, x):
        self.val = x
        self.next = None
```

倒序打印链表：（python简单，C++需要用辅助栈）

```
class Solution:
    def reversePrint(self, head: ListNode) -> List[int]:
        l = []
        while head:
            l.append(head.val)
            head = head.next
        return l[::-1]
# 感觉这就是链表转成列表然后倒序打印了
```

二叉树：

```
class TreeNode:
    def __init__(self, x):
        self.val = x
        self.left = None
        self.right = None
```

- **前序遍历**：根、左、右
- **中序遍历**：左、根、右
- **后续遍历**：左、右、根

这三种遍历都有**递归**和**循环**两种实现方法

从上到下打印二叉树：宽度优先遍历，逐层访问节点。

- **二叉搜索树**：左<=根<=右
- **堆**：最大堆：根最大，最小堆：根最小
- **红黑树**：节点分为红和黑，并且保证根到叶子的最长路径不超过最短路径的2倍。

重建二叉树（给你一个前序遍历和中序遍历的结果，重建这个二叉树）：

1. 前序遍历序列的第一个数字就是根节点的值。拿这个值扫描中序遍历序列得到位置，则这个位置左边的就是左节点，右边的就是右节点。中序遍历序列中找到几个左节点的数，前序遍历的根节点右边就数几个数。
2. 递归下去。

```
class Solution:
    def buildTree(self, preorder: List[int], inorder: List[int]) -> TreeNode:
        def recur(root, left, right):
            if left > right: return                          # 递归终止
            node = TreeNode(preorder[root])                  # 建立根节点
            i = dic[preorder[root]]                          # 划分根节点、左子树、右
子树
            node.left = recur(root + 1, left, i - 1)         # 开启左子树递归
            node.right = recur(i - left + root + 1, i + 1, right) # 开启右子树递归
            return node                                      # 回溯返回根节点

        dic, preorder = {}, preorder
        for i in range(len(inorder)):
            dic[inorder[i]] = i
        return recur(0, 0, len(inorder) - 1)

# 作者：jyd
# 链接：https://leetcode-cn.com/problems/zhong-jian-er-cha-shu-lcof/solution/mian-shi-
ti-07-zhong-jian-er-cha-shu-di-gui-fa-qin/
# 来源：力扣（LeetCode）
# 著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。
```

栈和队列：

python用列表模拟栈，用`append`压栈，用`pop`出栈，约定只能调用这俩函数。

用两个栈实现队列：维护两个栈，`appendTail`时，先倒一下栈，再用`append`，最后再倒一下栈；
用`deleteHead`

时，直接`pop`。

递归和循环：

- **递归**：一个函数内部调用这个函数本身
- **循环**：设置初始条件和终止条件，重复执行某一部分程序

斐波那契数列：递归的解法（动态规划）：

- **状态定义**：设dp为一维数组，其中dp[i]的值代表斐波那契数列第i个数字 。
- **转移方程**：dp[i+1]=dp[i]+dp[i−1] 。
- **初始状态**：dp[0]=0, dp[1]=1。
- **返回值**：dp[n]，即斐波那契数列的第n个数字。

由于dp列表第i项只与第i-1和第i-2项有关，因此只需要初始化三个整形变量sum, a, b，利用辅助变量sum使a, b两数字交替前进即可。

```
 def fib(self, n: int) -> int:
    a, b = 0, 1
    for _ in range(n):
        a, b = b, a + b
    return a % 1000000007

# 作者：jyd
# 链接：https://leetcode-cn.com/problems/fei-bo-na-qi-shu-lie-lcof/solution/mian-shi-ti-10-i-fei-bo-na-qi-shu-lie-dong-tai-gui/
# 来源：力扣（LeetCode）
# 著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。
```

青蛙跳台阶问题：分析以后才能知道这其实是斐波那契数列的变体，一般来说，类似于“有多少可能性”的问题都带有一定的递推性质。

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

1. 最后一步是跳上1级台阶时，之前剩下n-1个台阶，有f(n-1)种跳法。
2. 最后一步是跳上2级台阶时，之前剩下n-2个台阶，有f(n-2)种跳法。

而f(n)=f(n-1)+f(n-2)，于是这就是一个和斐波那契数列一样的递推公式，只是初始值不一样。

动态规划题都有四个固定的模块，**状态定义**、**状态转移方程**、**初始状态**、**返回值**。

查找与排序：

- 顺序查找
- 二分法查找
- 哈希表查找
- 二叉排序树查找

如果面试的时候，要求在排序或者部分排序的数组上找一个数或者统计一个数出现的次数，则可以尝试二分法。哈希表和二叉排序树法重点考察数据结构而不是算法。哈希表的时间复杂度是O(1)，但是需要额外的空间建立哈希表。

- 插入排序

- 冒泡排序
- 归并排序
- 快速排序

要熟记各种排序法的特点。

快速排序：在数组中选择一个数字，接下来把数组分成两部分，比选择的数字小的移到数组的左边，比选择的数字大的移到数组的右边。

第1个操作就是partition（切分），简单介绍如下：

partition（切分）操作，使得：

- 对于某个索引j，nums[j]已经排定，即nums[j]经过partition（切分）操作以后会放置在它“最终应该放置的地方”；
- nums[left]到nums[j-1]中的所有元素都不大于nums[j]；
- nums[j+1]到nums[right]中的所有元素都不小于nums[j]。

为避免最坏复杂度的情况，最好先随机选择一个元素。

步骤如下：

- 挑选基准值：从数列中挑出一个元素，称为"基准"（pivot）；
- 分割：重新排序数列，所有比基准值小的元素摆放在基准前面，所有比基准值大的元素摆在基准后面（与基准值相等的数可以到任何一边）。在这个分割结束之后，对基准值的排序就已经完成;
- 递归排序子序列：递归地将小于基准值元素的子序列和大于基准值元素的子序列排序。

对一个长度为n的数组排序，只需把`start`设为0、把`end`设为n-1,调用函数`QuickSort`即可。

下面是切分函数以及递归的快速排序代码：

```
def partition(arr,low,high):
    i = (low - 1)         # 最小元素索引
    pivot = arr[high]
    for j in range(low, high):

        # 当前元素小于或等于 pivot
        if arr[j] <= pivot:
            i = i + 1
            arr[i], arr[j] = arr[j], arr[i]

    arr[i+1], arr[high] = arr[high], arr[i+1]
    return (i + 1)

# arr[] --> 排序数组
# low  --> 起始索引
# high  --> 结束索引

# 快速排序函数
def quickSort(arr, low, high):
    if low < high:
        pi = partition(arr, low, high)
```

```
        quickSort(arr, low, pi - 1)
        quickSort(arr, pi + 1, high)

arr = [10, 7, 8, 9, 1, 5]
n = len(arr)
quickSort(arr, 0, n-1)
print ("排序后的数组:")
for i in range(n):
    print ("%d" %arr[i]),
```

partition（切分）操作总能排定一个元素，还能够知道这个元素它最终所在的位置，这样每经过一次partition（切分）操作就能缩小搜索的范围，这样的思想叫做“减而治之”（是“分治”思想的特例）

旋转数组的最小数字：因为旋转中的数组可以看作是两个排序的子数组（部分排序），**不过在python里，我只要搜索到断序的部分就行了。**

```
def minArray(self, numbers: List[int]) -> int:
    for i in range(len(numbers)-1):
        if numbers[i] > numbers[i+1]:
            return numbers[i+1]
        else:
            continue
    return numbers[0]
```

回溯法：

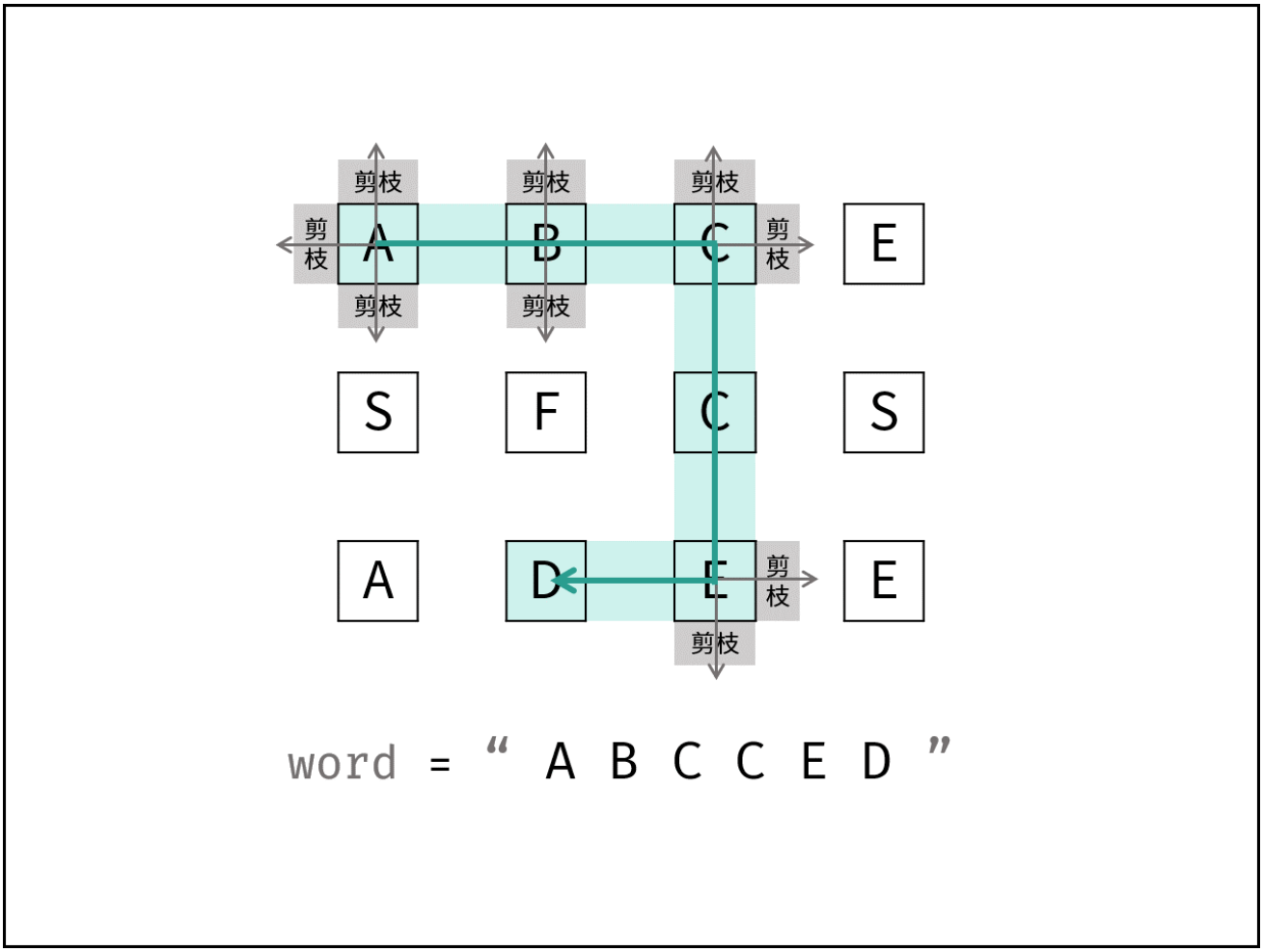
可以看作蛮力法的升级版。用回溯法解决的问题的所有选项可以形象地用树状结构表示。在某步有n个可能的选项，那么该步骤可以看成是树状结构中的一个节点，每个选项看成树中节点连接线，经过这些连接线到达该节点的n个子节点。树的叶节点对应着终结状态。如果在叶节点的状态满足题目的约条件，那么我们找到了一个可行的解决方案。

如果在叶节点的状态不满足约束条件，那么只好回溯到它的上一个节点再尝试其他的选项。如果上一个节点所有可能的选项都已经试过，并且不能到达满足约条件的终结状态，则再次回溯到上一个节点。如果所有节点的所有选项都已经尝试过仍然不能到达满足约束条件的终结状态，则该问题无解。

矩阵中的路径：

深度优先搜索（DFS）+剪枝：

- **深度优先搜索**：可以理解为暴力法遍历矩阵中所有字符串可能性。DFS通过递归，先朝一个方向搜到底，再回溯至上个节点，沿另一个方向搜索，以此类推。
- **剪枝**：在搜索中，遇到「这条路不可能和目标字符串匹配成功」的情况（例如：此矩阵元素和目标字符不同、此元素已被访问），则应立即返回，称之为「可行性剪枝」。



DFS解析：

- 递归参数：当前元素在矩阵board中的行列索引i和j，当前目标字符在word中的索引k。
- 终止条件：
  1. 返回false：(1)行或列索引越界，或(2)当前矩阵元素与目标字符不同，或(3)当前矩阵元素已访问过。（(3)可合并至(2)）。
  2. 返回true：k=len(word)-1，即字符串word已全部匹配。
- 递推工作：
  1. 标记当前矩阵元素：将`board[i][j]`修改为空字符`''`，代表此元素已访问过，防止之后搜索时重复访问。
  2. 搜索下一单元格：朝当前元素的上、下、左、右四个方向开启下层递归，使用`或`连接（代表只需找到一条可行路径就直接返回，不再做后续DFS），并记录结果至`res`。
     还原当前矩阵元素：将`board[i][j]`元素还原至初始值，即`word[k]`。
  3. 返回值：返回布尔量`res`，代表是否搜索到目标字符串。

```
def exist(self, board: List[List[str]], word: str) -> bool:
    def dfs(i, j, k):
        if not 0 <= i < len(board) or not 0 <= j < len(board[0]) or board[i][j] !=
word[k]: return False
        if k == len(word) - 1: return True
        board[i][j] = ''
        res = dfs(i + 1, j, k + 1) or dfs(i - 1, j, k + 1) or dfs(i, j + 1, k + 1) or
dfs(i, j - 1, k + 1)
```

```
            board[i][j] = word[k]
            return res

        for i in range(len(board)):
            for j in range(len(board[0])):
                if dfs(i, j, 0): return True
        return False

# 作者：jyd
# 链接：https://leetcode-cn.com/problems/ju-zhen-zhong-de-lu-jing-lcof/solution/mian-shi-ti-12-ju-zhen-zhong-de-lu-jing-shen-du-yo/
# 来源：力扣（LeetCode）
# 著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。
```

机器人的运动范围：

官方解之BFS：

数字数位之和：我们只需要对数x每次对10取余，就能知道数x的个位数是多少，然后再将x除10，这个操作等价于将x的十进制数向右移一位，删除个位数（类似于二进制中的`>>`右移运算符），不断重复直到x为0时结束。

```
def digitsum(n):
    ans = 0
    while n:
        ans += n % 10
        n //= 10
    return ans

# 作者：LeetCode-Solution
# 链接：https://leetcode-cn.com/problems/ji-qi-ren-de-yun-dong-fan-wei-lcof/solution/ji-qi-ren-de-yun-dong-fan-wei-by-leetcode-solution/
# 来源：力扣（LeetCode）
# 著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。
```

这道题还有一个隐藏的优化：我们在搜索的过程中搜索方向可以缩减为向右和向下，而不必再向上和向左进行搜索。

```
class Solution:
    def movingCount(self, m: int, n: int, k: int) -> int:
        from queue import Queue
        q = Queue()
        q.put((0, 0))
        s = set()
        while not q.empty():
            x, y = q.get()
            if (x, y) not in s and 0 <= x < m and 0 <= y < n and digitsum(x) +
digitsum(y) <= k:
                s.add((x, y))
                for nx, ny in [(x + 1, y), (x, y + 1)]:
```

```
                    q.put((nx, ny))
        return len(s)

# 作者：LeetCode-Solution
# 链接：https://leetcode-cn.com/problems/ji-qi-ren-de-yun-dong-fan-wei-lcof/solution/ji-qi-ren-de-yun-dong-fan-wei-by-leetcode-solution/
# 来源：力扣（LeetCode）
# 著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。
```

这里面有一个包`queue`，表示队列，这里用到了两个方法`queue.Queue.put()`，`queue.Queue.get()`。

另一个包`collections.deque`也有类似的功能，里面有`deque.append()`，`deque.appendleft()`，`deque.pop()`，`deque.popleft()`。

官方解之递推：

考虑到之前的方法提到搜索的方向只需要朝下或朝右，我们可以得出一种递推的求解方法。

定义`vis[i][j]`为(i,j)坐标是否可达，如果可达返回1，否则返回0。

首先(i,j)本身需要可以进入，因此需要先判断i和j的数位之和是否大于k，如果大于的话直接设置`vis[i][j]`为不可达即可。

否则，前面提到搜索方向只需朝下或朝右，因此(i,j)的格子只会从(i-1,j)或者(i,j-1)两个格子走过来（不考虑边界条件），那么`vis[i][j]`是否可达的状态则可由如下公式计算得到：

$$vis[i][j] = vis[i-1][j] \text{ or } vis[i][j-1]$$

即只要有一个格子可达，那么(i,j)这个格子就是可达的，因此我们只要遍历所有格子，递推计算出它们是否可达然后用变量`ans`记录可达的格子数量即可。

初始条件`vis[i][j]`=1，递推计算的过程中注意边界的处理。

```
def digitsum(n):
    ans = 0
    while n:
        ans += n % 10
        n //= 10
    return ans

class Solution:
    def movingCount(self, m: int, n: int, k: int) -> int:
        vis = set([(0, 0)])
        for i in range(m):
            for j in range(n):
                if ((i - 1, j) in vis or (i, j - 1) in vis) and digitsum(i) +
digitsum(j) <= k:
                    vis.add((i, j))
        return len(vis)

# 作者：LeetCode-Solution
```

```
# 链接：https://leetcode-cn.com/problems/ji-qi-ren-de-yun-dong-fan-wei-lcof/solution/ji-qi-ren-de-yun-dong-fan-wei-by-leetcode-solution/
# 来源：力扣（LeetCode）
# 著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。
```

动态规划与贪婪算法：

问题可以分解为子问题，每一个子问题都可以存在最优解，如果这些子问题的最优解组合起来能够得到整个问题的最优解，就可以用**动态规划**来解决问题。

剪绳子：长度为n的绳子，剪成m份，mn都是正整数，求最大乘积。

剪一刀的时候，我们有n-1种可能的选择，此时f(x)=max(f(i)*f(n-i))，从上到下递归。

动态规划解：

1. 我们想要求长度为n的绳子剪掉后的最大乘积，可以从前面比n小的绳子转移而来
2. 用一个dp数组记录从0到n长度的绳子剪掉后的最大乘积，也就是dp[i]表示长度i的绳子剪成m段后的最大乘积，初始化dp[2] = 1
3. 我们先把绳子剪掉第一段（长度为j），如果只剪掉长度为1，对最后的乘积无任何增益，所以从长度为2开始剪
4. 剪了第一段后，剩下(i - j)长度可以剪也可以不剪。如果不剪的话长度乘积即为j * (i - j)；如果剪的话长度乘积即为j * dp[i - j]。取两者最大值max(j * (i - j), j * dp[i - j])
5. 第一段长度j可以取的区间为[2,i)，对所有j不同的情况取最大值，因此最终dp[i]的转移方程为 `dp[i] = max(dp[i], max(j * (i - j), j * dp[i - j]))`
6. 最后返回dp[n]即可

```python
class Solution:
    def cuttingRope(self, n: int) -> int:
        dp = [0] * (n + 1)
        dp[2] = 1
        for i in range(3, n + 1):
            for j in range(2, i):
                dp[i] = max(dp[i], max(j * (i - j), j * dp[i - j]))
        return dp[n]

# 作者：edelweisskoko
# 链接：https://leetcode-cn.com/problems/jian-sheng-zi-lcof/solution/jian-zhi-offer-14-i-jian-sheng-zi-huan-s-xopj/
# 来源：力扣（LeetCode）
# 著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。
```

贪心算法解：

一般来说，可以证明，**当n≥5时，我们尽可能地多剪长度为3的绳子，当剩下的绳子长度为4时，剪成两段为2的绳子。**

1. 最优：3。把绳子尽可能切为多个长度为3的片段，留下的最后一段绳子的长度可能为0,1,2三种情况。
2. 次优：2 。若最后一段绳子长度为2；则保留，不再拆为1+1。
3. 最差：1。若最后一段绳子长度为1；则应把一份3 + 1替换为2 + 2，因为2×2>3×1。

剪绳子-II：涉及**大数越界下的求余问题**。

```
# 求 (x^a) % p —— 循环求余法
def remainder(x, a, p):
    rem = 1
    for _ in range(a):
        rem = (rem * x) % p
    return rem

# 求 (x^a) % p —— 快速幂求余
def remainder(x, a, p):
    rem = 1
    while a > 0:
        if a % 2: rem = (rem * x) % p
        x = x ** 2 % p
        a //= 2
    return rem

# 作者：jyd
# 链接：https://leetcode-cn.com/problems/jian-sheng-zi-ii-lcof/solution/mian-shi-ti-14-ii-jian-sheng-zi-iitan-xin-er-fen-f/
# 来源：力扣（LeetCode）
# 著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。
```

```
class Solution:
    def cuttingRope(self, n: int) -> int:
        if n <= 3:return n-1
        a, b, p, x= n // 3 - 1, n % 3, 1000000007, 3
        rem = remainder(x, a, p)
        if b == 0: return (rem * 3) % p
        if b == 1: return (rem * 4) % p
        return (rem * 6) % p

# 作者：jyd
# 链接：https://leetcode-cn.com/problems/jian-sheng-zi-ii-lcof/solution/mian-shi-ti-14-ii-jian-sheng-zi-iitan-xin-er-fen-f/
# 来源：力扣（LeetCode）
# 著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。
```

位运算：

二进制中1的个数：这个Python就一行代码

高质量的代码

数值的整数次方：（不用pow()和**）

```
# 自己运行没问题，但是提交会有TypeError
class Solution:
    def myPow(self, x: float, n: int) -> float:
        if n < 0:
            x = 1 / x
            n = - n
        res = 1
        while n:
            if n & 1:
                res *= x
            n >>= 1
            x *= x

# 作者：edelweisskoko
# 链接：https://leetcode-cn.com/problems/shu-zhi-de-zheng-shu-ci-fang-
lcof/solution/jian-zhi-offer-16-shu-zhi-de-zheng-shu-c-rgqy/
# 来源：力扣（LeetCode）
# 著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。
```

除了可以用迭代外，还可以用**递归法**。递归是比较好理解的：

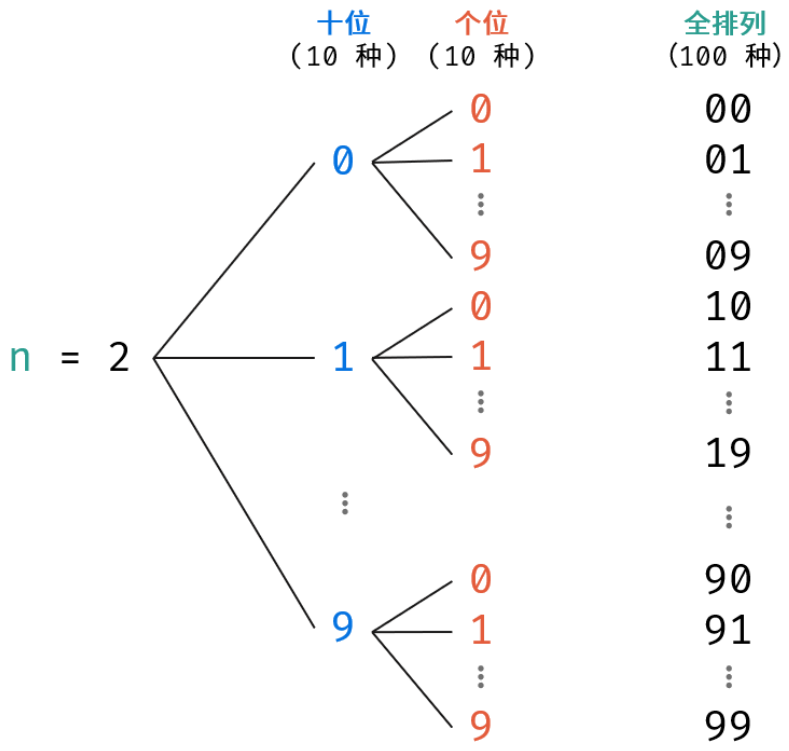
1. 如果n == 0，返回1
2. 如果n < 0，最终结果为 $1/x^{-n}$
3. 如果n为奇数，最终结果为$x * x^{n-1}$
4. 如果n为偶数，最终结果为 $x^{2*(n/2)}$

```
class Solution:
    def myPow(self, x: float, n: int) -> float:
        if n == 0:
            return 1
        elif n < 0:
            return 1/self.myPow(x, -n)
        elif n & 1:
            return x * self.myPow(x, n - 1)
        else:
            return self.myPow(x*x, n // 2)

# 作者：edelweisskoko
# 链接：https://leetcode-cn.com/problems/shu-zhi-de-zheng-shu-ci-fang-
lcof/solution/jian-zhi-offer-16-shu-zhi-de-zheng-shu-c-rgqy/
# 来源：力扣（LeetCode）
# 著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。
```

打印从1到最大的n位数：这个Python问题不大，但本题的主要考点是大数越界情况下的打印

如果是一个大数，假设long类型都不够表示，我们假设用字符串类型，这样也可以避开进位问题直接生成list[str]

使用基于分治的思想，固定高位，向低位递归。当个位被固定时，添加数字的字符串。



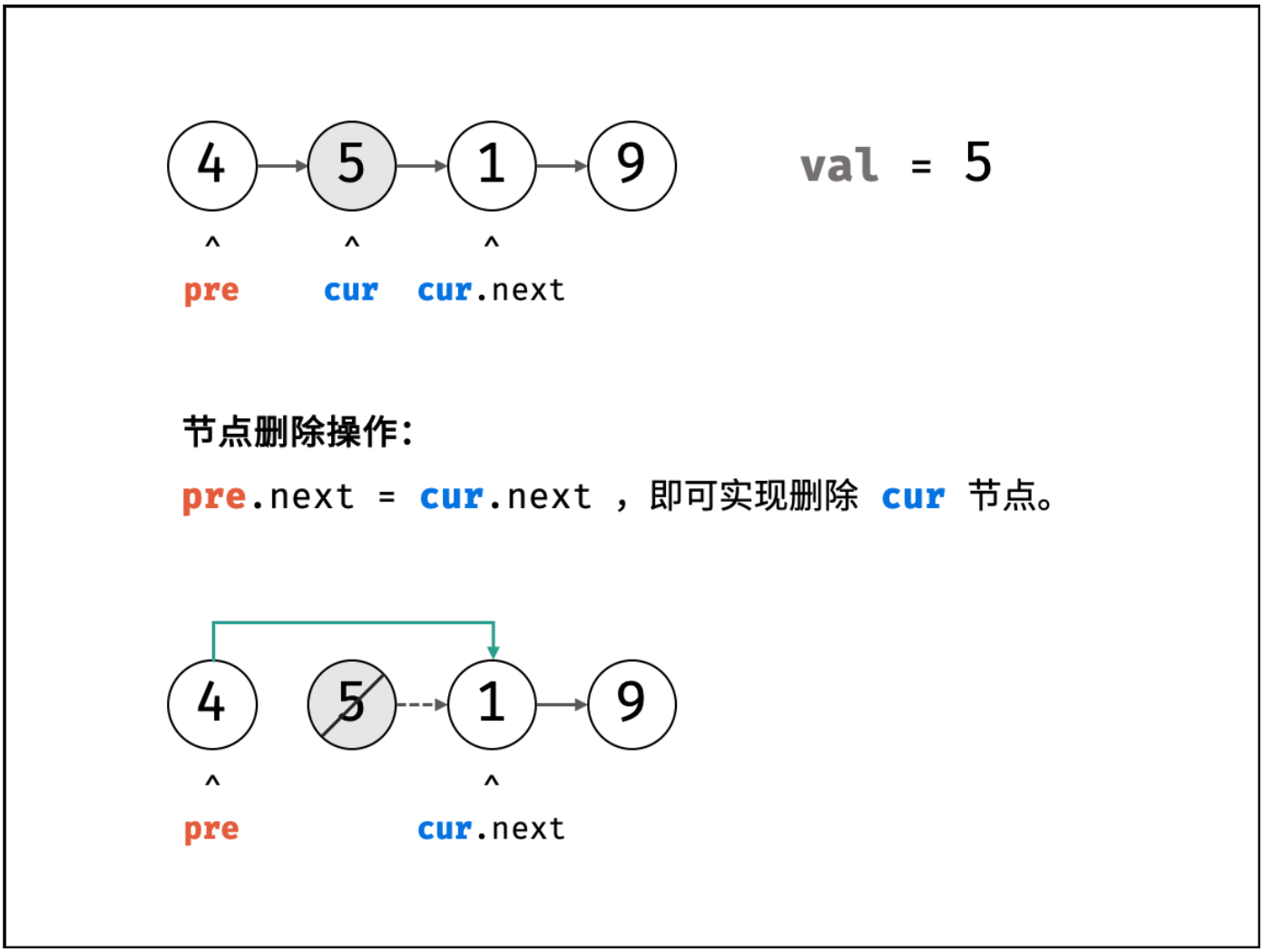
- 从 0 开始计数，全排列方案共有 10^n 种。
- 若从 1 开始计数，则方案共有 10^n-1 种。

```python
class Solution:
    def printNumbers(self, n: int) -> [int]:
        def dfs(x):
            if x == n: # 终止条件：已固定完所有位
                res.append(int(''.join(num))) # 拼接 num ，转成int，并添加至 res 尾部
                return
            for i in range(10): # 遍历 0 - 9
                num[x] = str(i) # 固定第 x 位为 i
                dfs(x + 1) # 开启固定第 x + 1 位

        num = ['0'] * n # 起始数字定义为 n 个 0 组成的字符列表
        res = [] # 数字字符串列表
        dfs(0) # 开启全排列递归
        del res[0] # 把0这一项去掉
        return res  # 拼接所有数字字符串，使用逗号隔开，并返回

# 作者：jyd
```

```
# 链接：https://leetcode-cn.com/problems/da-yin-cong-1dao-zui-da-de-nwei-shu-lcof/solution/mian-shi-ti-17-da-yin-cong-1-dao-zui-da-de-n-wei-2/
# 来源：力扣（LeetCode）
# 著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。
```

删除链表的节点：（在O(1)时间内，假设链表里一定有这个节点）

如果是顺序查找，那复杂度就是O(n)，我们删除节点i，先把下一个节点j内容复制到i，然后再把指针指向j的下一个节点，再删除节点j，效果相当于把i删除了。如果要删除的节点是尾节点，没有下一个节点，就还是顺序遍历了，如果链表就一个节点，那么还得删完以后把头节点设置为`null`。这样总的平均时间复杂度是O(1)（这都什么鬼。。）



```python
class Solution:
    def deleteNode(self, head: ListNode, val: int) -> ListNode:
        if head.val == val: return head.next
        pre, cur = head, head.next # 双节点
        while cur and cur.val != val:
            pre, cur = cur, cur.next
        if cur: pre.next = cur.next
        return head

# 作者：jyd
# 链接：https://leetcode-cn.com/problems/shan-chu-lian-biao-de-jie-dian-lcof/solution/mian-shi-ti-18-shan-chu-lian-biao-de-jie-dian-sh-2/
# 来源：力扣（LeetCode）
# 著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。
```

正则表达式匹配:

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

书上说可以用有限状态机，力扣官方用的动态规划，也有ACM大佬使用递归。

官方解：题目中的匹配是一个「逐步匹配」的过程：我们每次从字符串 pp 中取出一个字符或者「字符 + 星号」的组合，并在s中进行匹配。对于p中一个字符而言，它只能在s中匹配一个字符，匹配的方法具有唯一性；而对于p中字符 + 星号的组合而言，它可以在s中匹配任意自然数个字符，并不具有唯一性。因此我们可以考虑使用动态规划，对匹配的方案进行枚举。

我们用f[i][j]表示s的前i个字符与p中的前j个字符是否能够匹配。在进行状态转移时，我们考虑p的第j个字符的匹配情况：

- 如果p的第j个字符是一个小写字母，那么我们必须在s中匹配一个相同的小写字母，即

  f[i][j]=f[i-1][j-1] if s[i]=p[j], f[i][j]=False if s[i]!=p[j]

  也就是说，如果s的第i个字符与p的第j个字符不相同，那么无法进行匹配；否则我们可以匹配两个字符串的最后一个字符，完整的匹配结果取决于两个字符串前面的部分。

- 如果p的第j个字符是 `*`，那么就表示我们可以对p的第 j-1个字符匹配任意自然数次。我们有

  f[i][j]=f[i][j-2] if s[i]!=p[j-1], f[i][j]=f[i-1][j] or f[i][j-2]

- 在任意情况下，只要p[j]是 `.`，那么 p[j] 一定成功匹配s中的任意一个小写字母。

动态规划的边界条件为f[0][0]=True

```python
class Solution:
    def isMatch(self, s: str, p: str) -> bool:
        m, n = len(s), len(p)

        def matches(i: int, j: int) -> bool:
            if i == 0:
                return False
```

```
            if p[j - 1] == '.':
                return True
            return s[i - 1] == p[j - 1]

        f = [[False] * (n + 1) for _ in range(m + 1)]
        f[0][0] = True
        for i in range(m + 1):
            for j in range(1, n + 1):
                if p[j - 1] == '*':
                    f[i][j] |= f[i][j - 2]
                    if matches(i, j - 1):
                        f[i][j] |= f[i - 1][j]
                else:
                    if matches(i, j):
                        f[i][j] |= f[i - 1][j - 1]
        return f[m][n]

# 作者：LeetCode-Solution
# 链接：https://leetcode-cn.com/problems/zheng-ze-biao-da-shi-pi-pei-
lcof/solution/zheng-ze-biao-da-shi-pi-pei-by-leetcode-s3jgn/
# 来源：力扣（LeetCode）
# 著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。
```

表示数值的字符串：这道题一看就知道是用有限状态自动机解决的。

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

有一个特殊的状态，被称作「初始状态」。
还有一系列状态被称为「接受状态」，它们组成了一个特殊的集合。其中，一个状态可能既是「初始状态」，也是「接受状态」。
起初，这个自动机处于「初始状态」。随后，它顺序地读取字符串中的每一个字符，并根据当前状态和读入的字符，按照某个事先约定好的「转移规则」，从当前状态转移到下一个状态；当状态转移完成后，它就读取下一个字符。当字符串全部读取完毕后，如果自动机处于某个「接受状态」，则判定该字符串「被接受」；否则，判定该字符串「被拒绝」。

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

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

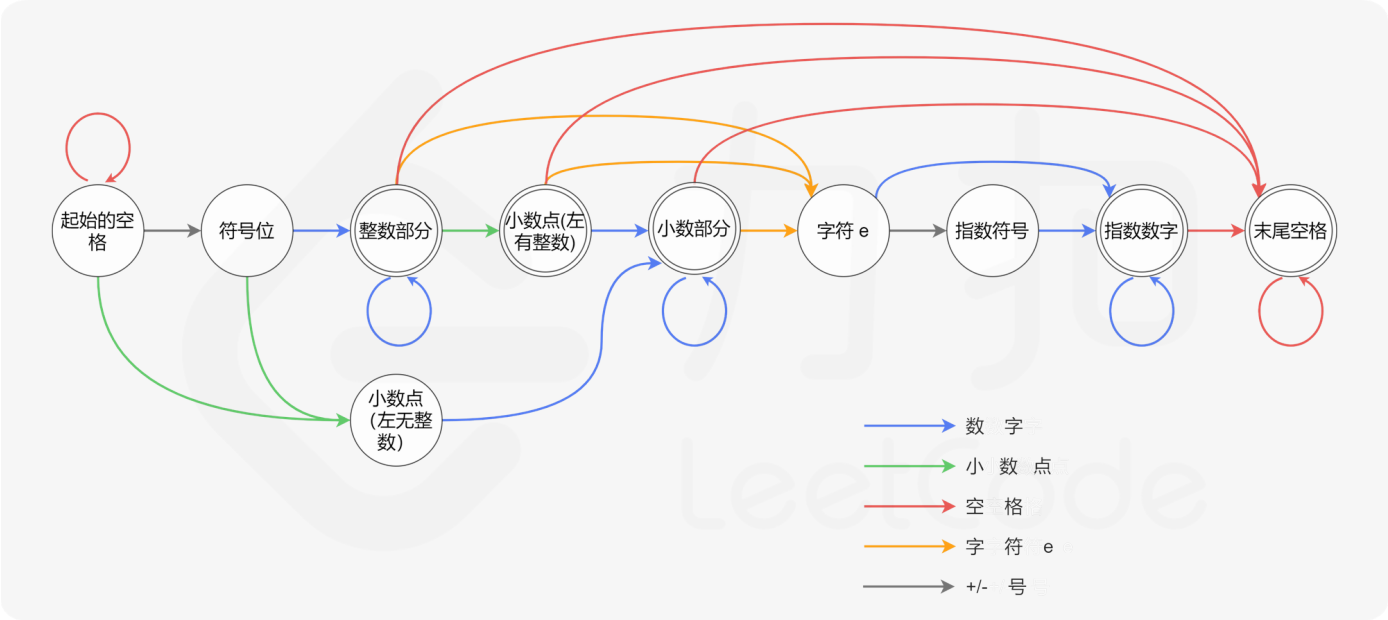
自动机驱动的编程，可以被看做一种暴力枚举方法的延伸：它穷尽了在任何一种情况下，对应任何的输入，需要做的事情。

自动机很像数电的状态转移图。首先列举所有状态：

1. 起始的空格
2. 符号位
3. 整数部分
4. 左侧有整数的小数点
5. 左侧无整数的小数点（根据前面的第二条额外规则，需要对左侧有无整数的两种小数点做区分）

6. 小数部分
7. 字符 \text{e}e
8. 指数部分的符号位
9. 指数部分的整数部分
10. 末尾的空格

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



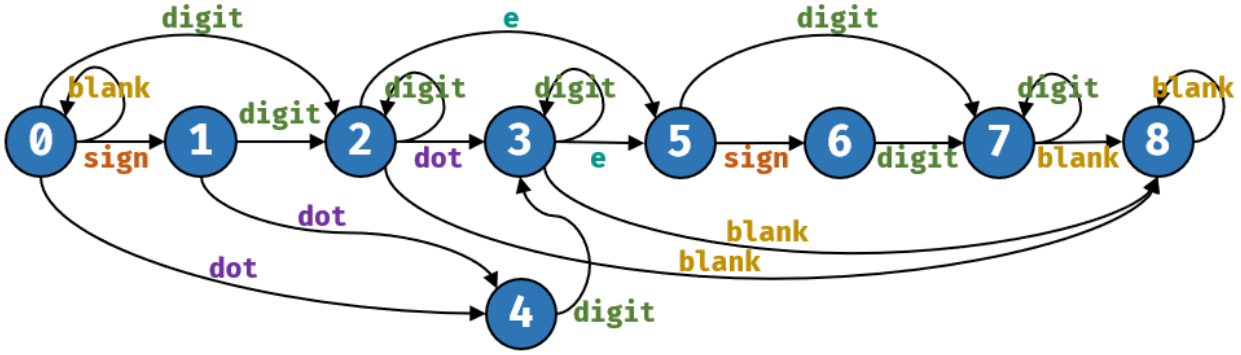

```python
from enum import Enum

class Solution:
    def isNumber(self, s: str) -> bool:
        State = Enum("State", [
            "STATE_INITIAL",
            "STATE_INT_SIGN",
            "STATE_INTEGER",
            "STATE_POINT",
            "STATE_POINT_WITHOUT_INT",
            "STATE_FRACTION",
            "STATE_EXP",
            "STATE_EXP_SIGN",
            "STATE_EXP_NUMBER",
            "STATE_END",
        ])
        Chartype = Enum("Chartype", [
            "CHAR_NUMBER",
            "CHAR_EXP",
            "CHAR_POINT",
            "CHAR_SIGN",
            "CHAR_SPACE",
            "CHAR_ILLEGAL",
```

```
        ])

        def toChartype(ch: str) -> Chartype:
            if ch.isdigit():
                return Chartype.CHAR_NUMBER
            elif ch.lower() == "e":
                return Chartype.CHAR_EXP
            elif ch == ".":
                return Chartype.CHAR_POINT
            elif ch == "+" or ch == "-":
                return Chartype.CHAR_SIGN
            elif ch == " ":
                return Chartype.CHAR_SPACE
            else:
                return Chartype.CHAR_ILLEGAL

        transfer = {
            State.STATE_INITIAL: {
                Chartype.CHAR_SPACE: State.STATE_INITIAL,
                Chartype.CHAR_NUMBER: State.STATE_INTEGER,
                Chartype.CHAR_POINT: State.STATE_POINT_WITHOUT_INT,
                Chartype.CHAR_SIGN: State.STATE_INT_SIGN,
            },
            State.STATE_INT_SIGN: {
                Chartype.CHAR_NUMBER: State.STATE_INTEGER,
                Chartype.CHAR_POINT: State.STATE_POINT_WITHOUT_INT,
            },
            State.STATE_INTEGER: {
                Chartype.CHAR_NUMBER: State.STATE_INTEGER,
                Chartype.CHAR_EXP: State.STATE_EXP,
                Chartype.CHAR_POINT: State.STATE_POINT,
                Chartype.CHAR_SPACE: State.STATE_END,
            },
            State.STATE_POINT: {
                Chartype.CHAR_NUMBER: State.STATE_FRACTION,
                Chartype.CHAR_EXP: State.STATE_EXP,
                Chartype.CHAR_SPACE: State.STATE_END,
            },
            State.STATE_POINT_WITHOUT_INT: {
                Chartype.CHAR_NUMBER: State.STATE_FRACTION,
            },
            State.STATE_FRACTION: {
                Chartype.CHAR_NUMBER: State.STATE_FRACTION,
                Chartype.CHAR_EXP: State.STATE_EXP,
                Chartype.CHAR_SPACE: State.STATE_END,
            },
            State.STATE_EXP: {
                Chartype.CHAR_NUMBER: State.STATE_EXP_NUMBER,
                Chartype.CHAR_SIGN: State.STATE_EXP_SIGN,
```

```
            },
            State.STATE_EXP_SIGN: {
                Chartype.CHAR_NUMBER: State.STATE_EXP_NUMBER,
            },
            State.STATE_EXP_NUMBER: {
                Chartype.CHAR_NUMBER: State.STATE_EXP_NUMBER,
                Chartype.CHAR_SPACE: State.STATE_END,
            },
            State.STATE_END: {
                Chartype.CHAR_SPACE: State.STATE_END,
            },
        }

        st = State.STATE_INITIAL
        for ch in s:
            typ = toChartype(ch)
            if typ not in transfer[st]:
                return False
            st = transfer[st][typ]

        return st in [State.STATE_INTEGER, State.STATE_POINT, State.STATE_FRACTION,
State.STATE_EXP_NUMBER, State.STATE_END]

# 作者：LeetCode-Solution
# 链接：https://leetcode-cn.com/problems/biao-shi-shu-zhi-de-zi-fu-chuan-
lcof/solution/biao-shi-shu-zhi-de-zi-fu-chuan-by-leetcode-soluti/
# 来源：力扣（LeetCode）
# 著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。
```

如果不用emun，也不想表示太复杂，可以换一种写法。但还是用dict来表示状态表。

状态转移图



表示	缩写	含义	字符
blank		空格	' '
sign	s	正负号	'+','-'
digit	d	数字	'0'~'9'
dot	.	小数点	'.'
e	e	幂符号	'e','E'

状态	描述
0	起始的 blank
1	e 之前的 sign
2	dot 之前的 digit
3	dot 之后的 digit
4	当 dot 前为空时，dot 后的 digit
5	e
6	e 之后的 sign
7	e 之后的 digit
8	尾部的 blank

```
class Solution:
    def isNumber(self, s: str) -> bool:
        states = [
            { ' ': 0, 's': 1, 'd': 2, '.': 4 }, # 0. start with 'blank'
            { 'd': 2, '.': 4 } ,                # 1. 'sign' before 'e'
            { 'd': 2, '.': 3, 'e': 5, ' ': 8 }, # 2. 'digit' before 'dot'
            { 'd': 3, 'e': 5, ' ': 8 },         # 3. 'digit' after 'dot'
            { 'd': 3 },                         # 4. 'digit' after 'dot' (‘blank’
before 'dot')
            { 's': 6, 'd': 7 },                 # 5. 'e'
            { 'd': 7 },                         # 6. 'sign' after 'e'
            { 'd': 7, ' ': 8 },                 # 7. 'digit' after 'e'
            { ' ': 8 }                          # 8. end with 'blank'
        ]
        p = 0                           # start with state 0
        for c in s:
            if '0' <= c <= '9': t = 'd' # digit
            elif c in "+-": t = 's'     # sign
            elif c in "eE": t = 'e'     # e or E
            elif c in ". ": t = c       # dot, blank
            else: t = '?'               # unknown
            if t not in states[p]: return False
```

```
            p = states[p][t]
        return p in (2, 3, 7, 8)

# 作者：jyd
# 链接：https://leetcode-cn.com/problems/biao-shi-shu-zhi-de-zi-fu-chuan-lcof/solution/mian-shi-ti-20-biao-shi-shu-zhi-de-zi-fu-chuan-y-2/
# 来源：力扣（LeetCode）
# 著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。
```

调整数组顺序使奇数位于偶数前面：使用双指针法（首尾双指针和快慢双指针）

首尾双指针

- 定义头指针`left`，尾指针`right`
- `left一直往右移，直到它指向的值为偶数
- `right`一直往左移，直到它指向的值为奇数
- 交换`nums[left]`和`nums[right]`
- 重复上述操作，直到`left == right`

```
class Solution:
    def exchange(self, nums: List[int]) -> List[int]:
        left, right = 0, len(nums)-1
        while(left<right):
            if nums[left] % 2 != 0:
                left += 1
                continue
            if nums[right] % 2 == 0:
                right -= 1
                continue
            nums[left], nums[right] = nums[right], nums[left]
            left += 1
            right -= 1
        return nums
```

快慢双指针

- 定义快慢双指针`fast`和`low`，`fast`在前，`low`在后
- `fast`的作用是向前搜索奇数位置，`low`的作用是指向下一个奇数应当存放的位置
- `fast`向前移动，当它搜索到奇数时，将它和`nums[low]`交换，此时`low`向前移动一个位置
- 重复上述操作，直到`fast`指向数组末尾

```python
class Solution:
    def exchange(self, nums: List[int]) -> List[int]:
        low, fast = 0, 0
        while(fast < len(nums)):
            if nums[fast] % 2 != 0:
                nums[fast], nums[low] = nums[low], nums[fast]
                low += 1
            fast += 1
        return nums
```

代码的鲁棒性

链表中倒数第k个节点：这个还是使用双指针。

1. **初始化**： 前指针`former`、后指针`latter`，双指针都指向头节点`head`。
2. **构建双指针距离**：前指针`former`先向前走`k`步（结束后，双指针`former`和`latter`间相距`k`步）。
3. **双指针共同移动**：循环中，双指针`former`和`latter`每轮都向前走一步，直至`former`走过链表**尾节点**时跳出（跳出后，`latter`与尾节点距离为`k-1`，即`latter`指向倒数第`k`个节点）。
4. **返回值**：返回`latter`即可。

```python
class Solution:
    def getKthFromEnd(self, head: ListNode, k: int) -> ListNode:
        former, latter = head, head
        for _ in range(k):
            if not former: return
            former = former.next
        while former:
            former, latter = former.next, latter.next
        return latter

# 作者：jyd
# 链接：https://leetcode-cn.com/problems/lian-biao-zhong-dao-shu-di-kge-jie-dian-lcof/solution/mian-shi-ti-22-lian-biao-zhong-dao-shu-di-kge-j-11/
# 来源：力扣（LeetCode）
# 著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。
```

反转链表:

迭代:

```python
class Solution:
    def reverseList(self, head: ListNode) -> ListNode:
        pre, cur = None, head
        while cur:
            nxt = cur.next
            cur.next = pre
            pre = cur
            cur = nxt
```

```
        return pre

# 作者：edelweisskoko
# 链接：https://leetcode-cn.com/problems/fan-zhuan-lian-biao-lcof/solution/shi-pin-tu-jie-jian-zhi-offer-24-fan-zhu-oym7/
# 来源：力扣（LeetCode）
# 著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。
```

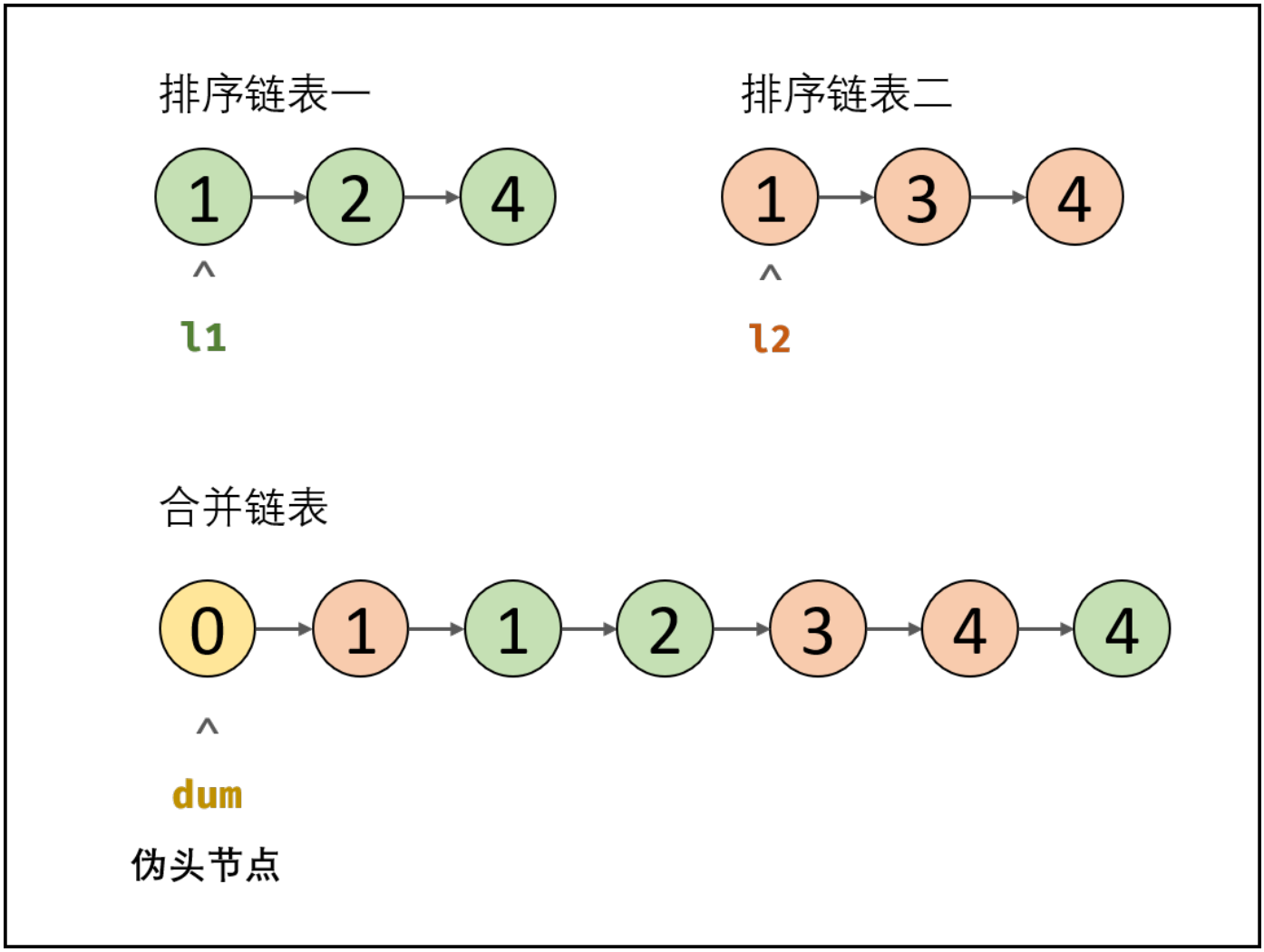
递归：

```
class Solution:
    def reverseList(self, head: ListNode) -> ListNode:
        if not head or not head.next:
            return head
        newHead = self.reverseList(head.next)
        head.next.next = head
        head.next = None
        return newHead

# 作者：edelweisskoko
# 链接：https://leetcode-cn.com/problems/fan-zhuan-lian-biao-lcof/solution/shi-pin-tu-jie-jian-zhi-offer-24-fan-zhu-oym7/
# 来源：力扣（LeetCode）
# 著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。
```

合并两个排序的列表：

两个链表是递增的，所以可以使用双指针l1, l2遍历，再根据l1.val和la.val的大小关系确定节点添加顺序，两节点交替前进，直至遍历完毕。

引入伪头节点：初始化一个辅助节点dum作为合并链表的伪头节点，将各节点添加到dum后



1. **初始化**： 伪头节点`dum` ， 节点 `cur` 指向 `dum` 。
2. **循环合并**： 当`l1`或 `l2`为空时跳出；
    1. 当`l1.val < l2.val` 时： `cur` 的后继节点指定为`l1` ， 并`l1` 向前走一步；
    2. 当`l1.val >= l2.val`时： `cur` 的后继节点指定为`l2` ， 并 `l2` 向前走一步 ；
    3. 节点 `cur` 向前走一步，即 `cur = cur.next`。
3. **合并剩余尾部**： 跳出时有两种情况，即`l1`为空 或`l2`为空。
    1. 若`l1 != null`： 将 `l1`添加至节点 `cur` 之后；
    2. 否则： 将 `l2` 添加至节点 `cur` 之后。
4. **返回值**： 合并链表在伪头节点 `dum` 之后，因此返回 `dum.next` 即可。

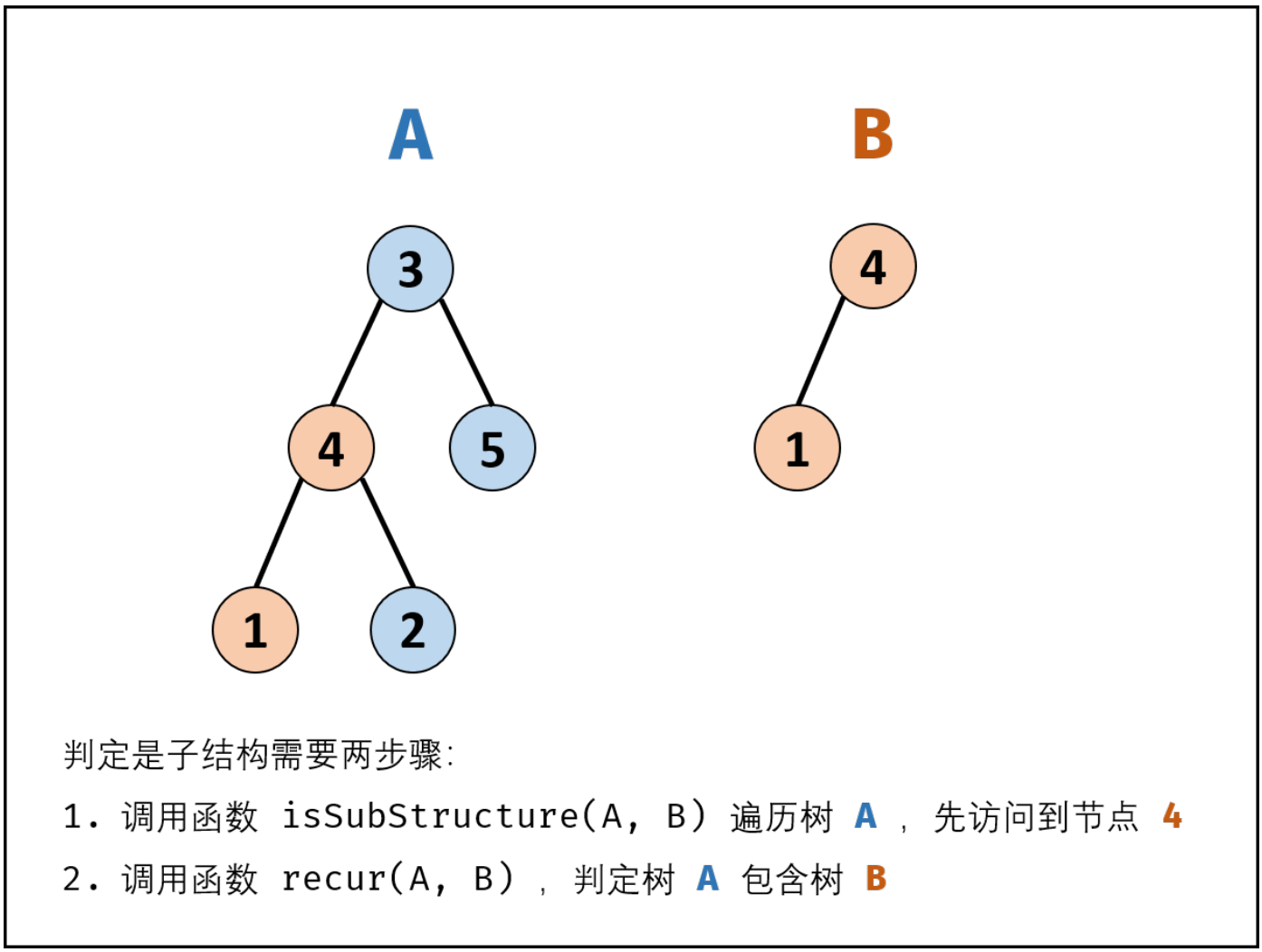
```python
class Solution:
    def mergeTwoLists(self, l1: ListNode, l2: ListNode) -> ListNode:
        cur = dum = ListNode(0)
        while l1 and l2:
            if l1.val < l2.val:
                cur.next, l1 = l1, l1.next
            else:
                cur.next, l2 = l2, l2.next
            cur = cur.next
        cur.next = l1 if l1 else l2
```

```
        return dum.next

# 作者：jyd
# 链接：https://leetcode-cn.com/problems/he-bing-liang-ge-pai-xu-de-lian-biao-lcof/solution/mian-shi-ti-25-he-bing-liang-ge-pai-xu-de-lian-b-2/
# 来源：力扣（LeetCode）
# 著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。
```

树的子结构：输入两颗二叉树A和B，判断B是不是A的子结构。

解法分成两步：

- **第一步**：在树A中找到和树B的根节点值一样的节点R；
- **第二步**：判断树A中与R为根节点的子树是不是包含和树B一样的结构。



recur(A, B) 函数：

终止条件：

- 当节点 `B` 为空：说明树 `B` 已匹配完成（越过叶子节点），因此返回 `true` ；
- 当节点 `A` 为空：说明已经越过树 `A` 叶子节点，即匹配失败，返回 `false` ；
- 当节点 `A` 和 `B` 的值不同：说明匹配失败，返回 `false` ；

返回值：

- 判断 `A` 和 `B` 的左子节点是否相等，即 `recur(A.left, B.left)` ；
- 判断 `A` 和 `B` 的右子节点是否相等，即 `recur(A.right, B.right)` ；

isSubStructure(A, B) 函数：

特例处理： 当 树 `A` 为空 或 树 `B` 为空 时，直接返回 `false` ；
返回值： 若树 `B` 是树 `A` 的子结构，则必满足以下三种情况之一，因此用或 `||` 连接；

- 以 节点 `A` 为根节点的子树 包含树 `B` ，对应 `recur(A, B)`；
- 树 `B` 是 树 `A` 左子树 的子结构，对应 `isSubStructure(A.left, B)`；
- 树 `B` 是 树 `A` 右子树 的子结构，对应 `isSubStructure(A.right, B)`；

```
class Solution:
    def isSubStructure(self, A: TreeNode, B: TreeNode) -> bool:
        def recur(A, B):
            if not B: return True
            if not A or A.val != B.val: return False
            return recur(A.left, B.left) and recur(A.right, B.right)

        return bool(A and B) and (recur(A, B) or self.isSubStructure(A.left, B) or
self.isSubStructure(A.right, B))

# 作者：jyd
# 链接：https://leetcode-cn.com/problems/shu-de-zi-jie-gou-lcof/solution/mian-shi-ti-26-
shu-de-zi-jie-gou-xian-xu-bian-li-p/
# 来源：力扣（LeetCode）
# 著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。
```

-TBC-