📚 E-Level 通关秘籍

核心模板 1：栈 (Stack) - 括号匹配

看到什么题目用这个?
题目里有 **括号 () [] {}**，要求判断是否 **“有效”、“匹配”、“合法”**。

```
    # 模板：括号匹配
# 适用于 LeetCode 20, OpenJudge 03704 等

def solve():
    # --- 你需要修改的部分：读取输入 ---
    s = input() # 假设输入只有一行字符串

    # --- 以下是固定模板，一般不需要修改 ---
    stack = []
    mapping = {')': '(', ']': '[', '}': '{'}

    for char in s:
        if char in mapping.values(): # 如果是左括号
            stack.append(char)
        elif char in mapping.keys(): # 如果是右括号
            # 如果栈是空的，或者栈顶的左括号不匹配
            if not stack or mapping[char] != stack.pop():
                # --- 你需要修改的部分：根据题目要求输出 ---
                print("无效") # 或者 print("false"), print("NO")
                return # 提前结束

    # --- 你需要修改的部分：根据题目要求输出 ---
    # 如果循环结束时栈是空的，说明匹配成功
    if not stack:
        print("有效") # 或者 print("true"), print("YES")
    else:
        # 如果栈不为空，说明有多余的左括号
        print("无效") # 或者 print("false"), print("NO")

# --- 主程序入口 ---
# solve() # 如果题目只有一个测试用例，直接调用
# 如果有多个测试用例，像这样写：
# n = int(input())
# for _ in range(n):
#     solve()
```

真题实战 1：LeetCode 20. 有效的括号

- **题目描述**：给定一个只包括 '('，')'，'{'，'}'，'['，']' 的字符串 s ，判断字符串是否有效。
- **输入**：s = "()[]{}"
- **输出**：true

```
    class Solution:
    def isValid(self, s: str) -> bool:
        # --- 这是固定模板 ---
```

```
        stack = []
        mapping = {')': '(', ']': '[', '}': '{'}

        for char in s:
            if char in mapping.values():
                stack.append(char)
            elif char in mapping.keys():
                if not stack or mapping[char] != stack.pop():
                    # 题目要求返回布尔值
                    return False

        # 题目要求返回布尔值
        return not stack
```

核心模板 2：队列 (Queue) - 模拟报数淘汰

看到什么题目用这个?
题目描述有 **“围成一圈”、“报数”、“淘汰”、“排队”** 等关键词。

```
    # 模板：约瑟夫问题/报数淘汰
# 适用于 OpenJudge 02746 等
from collections import deque

def solve():
    # --- 你需要修改的部分：读取输入 ---
    try:
        line = input()
        if not line: return
        n, m = map(int, line.split())
        # 检查结束条件
        if n == 0 and m == 0:
            return False # 返回False来停止循环
    except (ValueError, IndexError, EOFError):
        return False

    # --- 以下是固定模板，一般不需要修改 ---

    # 1. 初始化队列，放入 1 到 n
    queue = deque(range(1, n + 1))

    # 2. 循环直到只剩一个
    while len(queue) > 1:
        # 3. 报数 m-1 次：将队头元素移到队尾
        for _ in range(m - 1):
            queue.append(queue.popleft())

        # 4. 淘汰第 m 个元素（当前队头）
        queue.popleft()

    # 5. 剩下的最后一个就是答案
    winner = queue[0]

    # --- 你需要修改的部分：根据题目要求输出 ---
    print(winner)
```

```
    return True # 返回True表示继续处理下一组

# --- 主程序入口 ---
while solve():
    pass
```

核心模板 3：链表 - 反转与合并

看到什么题目用这个?
题目明确提到 **“链表 (Linked List)”**，并要求 **“反转”** 或 **“合并”**。

```
    # --- 完整代码模板：反转链表 ---

# 1. 定义链表节点类 (这是必须的)
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

# 2. 反转函数 (固定模板)
def reverse_list(head: ListNode) -> ListNode:
    prev = None
    curr = head
    while curr:
        next_temp = curr.next
        curr.next = prev
        prev = curr
        curr = next_temp
    return prev

# --- 如何在考试中使用 ---
# 考试时，你通常不需要自己创建链表，题目会给你一个 head。
# 你只需要实现 `reverse_list` 函数的主体部分。

# LeetCode上的完整类结构
class Solution:
    def reverseList(self, head: ListNode) -> ListNode:
        prev = None
        curr = head
        while curr:
            next_temp = curr.next
            curr.next = prev
            prev = curr
            curr = next_temp
        return prev
```

```
    # --- 完整代码模板：合并两个有序链表 ---

# 1. 定义链表节点类
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
```

```
        self.next = next

# 2. 合并函数 (固定模板)
def merge_two_lists(l1: ListNode, l2: ListNode) -> ListNode:
    # a. 创建一个哨兵节点，简化代码
    dummy = ListNode(-1)
    # b. 创建一个 tail 指针，用于构建新链表
    tail = dummy

    # c. 当两个链表都还有节点时
    while l1 and l2:
        if l1.val <= l2.val:
            tail.next = l1
            l1 = l1.next
        else:
            tail.next = l2
            l2 = l2.next
        tail = tail.next

    # d. 连接剩下的链表部分
    tail.next = l1 if l1 else l2

    # e. 返回哨兵节点的下一个节点，即新链表的头
    return dummy.next
```

核心模板 4：字典/集合 - 分组与计数

看到什么题目用这个?
题目要求 **统计频率**、**按类别分组**、**检查重复**。

```
    # --- 完整代码模板：分组计数 ---
from collections import defaultdict
import sys

def solve():
    # 使用 defaultdict(list)，当键不存在时，自动创建一个空列表
    groups = defaultdict(list)

    # --- 你需要修改的部分：读取输入 ---
    try:
        n = int(sys.stdin.readline())
        for _ in range(n):
            # 假设每行是 "key value"
            key, value = sys.stdin.readline().strip().split()
            # 将 value 加入到 key 对应的列表中
            groups[key].append(value)
    except (ValueError, IndexError):
        return

    # --- 你需要修改的部分：处理和输出 ---
    # 示例：按 key 排序，并打印每个组
    for key in sorted(groups.keys()):
        values = groups[key]
        print(f"{key}: {' '.join(values)}")
```

```
# --- 主程序入口 ---
# solve()
```

真题实战 2：OpenJudge 06640. 倒排索引

-
- **题目描述**：读取N篇文档，每篇文档包含若干单词。然后对M个查询单词，输出包含该单词的所有文档编号。
- **输入**：

```
    3
1 this is a sample
2 this is another example
3 example
2
this
example
```

- **输出**：

```
    1 2
2 3
```

```
    import sys
from collections import defaultdict

def solve():
    # --- 以下是固定模板 ---
    # 使用 defaultdict(list)，键是单词，值是文档编号列表
    inverted_index = defaultdict(list)

    # --- 你需要修改的部分：读取文档 ---
    try:
        n = int(sys.stdin.readline())
        for doc_id in range(1, n + 1):
            words = sys.stdin.readline().strip().split()
            # 第一个词是文档编号，忽略
            for word in words[1:]:
                # 避免重复添加同一个文档编号
                if doc_id not in inverted_index[word]:
                    inverted_index[word].append(doc_id)
    except (ValueError, IndexError):
        return

    # --- 你需要修改的部分：处理查询 ---
    try:
        m = int(sys.stdin.readline())
        for _ in range(m):
            query_word = sys.stdin.readline().strip()
```

```
            if query_word in inverted_index:
                # 题目要求从小到大输出，而我们是按顺序添加的，所以正好
                print(" ".join(map(str, inverted_index[query_word])))
            else:
                print("NOT FOUND")
    except (ValueError, IndexError):
        return

solve()
```

📚 第三部分：更多 E-Level 题目与完整代码

题目 4：E108. 将有序数组转换为二叉搜索树

题目描述：给你一个升序排列的整数数组 nums ，请你将其转换为一棵**高度平衡**的二叉搜索树。

思路：**递归 + 分治。**

1. 找到数组的 **中间元素**，它就是当前树（或子树）的根节点。
2. 中间元素 **左边** 的部分，递归地构建 **左子树。**
3. 中间元素 **右边** 的部分，递归地构建 **右子树。**
4. 当子数组为空时，返回 None。

```
    class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

class Solution:
    def sortedArrayToBST(self, nums: List[int]) -> Optional[TreeNode]:

        def build(left_index, right_index):
            # 递归终止条件：如果区间无效
            if left_index > right_index:
                return None

            # 1. 找到中间元素作为根
            mid_index = (left_index + right_index) // 2
            root = TreeNode(nums[mid_index])

            # 2. 递归构建左子树
            root.left = build(left_index, mid_index - 1)

            # 3. 递归构建右子树
            root.right = build(mid_index + 1, right_index)

            return root
```

```
        return build(0, len(nums) - 1)
```

题目 5：E23563 - 多项式时间复杂度

题目描述：给一个字符串形式的多项式，找出最高次项。

思路：字符串处理。

1. 按 + 分割字符串，得到各项。
2. 遍历每一项，用 split('n^') 或 find('n') 来解析出系数和指数。
3. 注意处理特殊情况，如 n (指数为1)、常数项 (指数为0)、n^... (系数为1)。
4. 只在系数不为0时，才更新 max_power。

```
    import sys

def solve():
    line = sys.stdin.readline().strip()
    if not line: return

    terms = line.split('+')
    max_power = 0

    for term in terms:
        coeff_str = ""
        power_str = ""

        if 'n' in term:
            parts = term.split('n')
            coeff_str = parts[0]

            if '^' in parts[1]:
                power_str = parts[1][1:] # 去掉 '^'
            else: # '5n' 这种情况
                power_str = "1"
        else: # 常数项
            coeff_str = term
            power_str = "0"

        coeff = int(coeff_str) if coeff_str else 1
        power = int(power_str) if power_str else 0

        if coeff != 0:
            max_power = max(max_power, power)

    print(f"n^{max_power}")

solve()
```