Assignment #4: Matrix, 链表 & Backtracking

Updated 2226 GMT+8 Sep 29, 2025

2025 fall, Complied by 林奕妃、环境科学与工程学院

> **说明：**
>
> 1. **解题与记录：**
>
>   对于每一个题目，请提供其解题思路（可选），并附上使用Python或C++编写的源代码（确保已在OpenJudge， Codeforces，LeetCode等平台上获得Accepted）。请将这些信息连同显示“Accepted”的截图一起填写到下方的作业模板中。（推荐使用Typora https://typoraio.cn 进行编辑，当然你也可以选择Word。）无论题目是否已通过，请标明每个题目大致花费的时间。
>
> 2. **提交安排：**提交时，请首先上传PDF格式的文件，并将.md或.doc格式的文件作为附件上传至右侧的“作业评论”区。确保你的Canvas账户有一个清晰可见的本人头像，提交的文件为PDF格式，并且“作业评论”区包含上传的.md或.doc附件。
>
> 3. **延迟提交：**如果你预计无法在截止日期前提交作业，请提前告知具体原因。这有助于我们了解情况并可能为你提供适当的延期或其他帮助。
>
> 请按照上述指导认真准备和提交作业，以保证顺利完成课程要求。

1. 题目

E18161: 矩阵运算（30分钟）

matrices, http://cs101.openjudge.cn/pctbook/E18161/

请使用@矩阵相乘运算符。

思路：Unable to import 'numpy' (import-error)，不能使用numpy库，则需要手动实现矩阵的读取、乘法和加法运算。进行运算之前要先判断两个矩阵的行列数是否符合运算规则，再按照要求进行运算。

代码：

```
import sys

class Matrix:
    def __init__(self, data):
        """
        初始化矩阵对象。
        data: 一个二维列表 (list of lists)。
        """
        self.data = data
        self.rows = len(data)
```

```
        self.cols = len(data[0]) if self.rows > 0 else 0

    def __repr__(self):
        """定义对象的字符串表示，方便调试。"""
        return f"Matrix({self.data})"

    def __matmul__(self, other):
        """
        定义 '@' 运算符的行为 (矩阵乘法)。
        self @ other
        """
        # 检查乘法条件
        if self.cols != other.rows:
            raise ValueError("Matrix dimensions are not compatible for
multiplication.")

        # 创建结果矩阵并初始化为 0
        result_data = [[0 for _ in range(other.cols)] for _ in range(self.rows)]

        # 执行乘法计算
        for i in range(self.rows):
            for j in range(other.cols):
                for k in range(self.cols):
                    result_data[i][j] += self.data[i][k] * other.data[k][j]

        # 返回一个新的 Matrix 对象
        return Matrix(result_data)

    def __add__(self, other):
        """
        定义 '+' 运算符的行为 (矩阵加法)。
        self + other
        """
        # 检查加法条件
        if self.rows != other.rows or self.cols != other.cols:
            raise ValueError("Matrix dimensions are not compatible for
addition.")

        # 创建结果矩阵
        result_data = [[0 for _ in range(self.cols)] for _ in range(self.rows)]

        # 执行加法计算
        for i in range(self.rows):
            for j in range(self.cols):
                result_data[i][j] = self.data[i][j] + other.data[i][j]

        # 返回一个新的 Matrix 对象
        return Matrix(result_data)

def read_matrix_as_object():
    """
    读取矩阵数据并返回一个 Matrix 对象。
    """
    rows, cols = map(int, sys.stdin.readline().split())
    matrix_data = [list(map(int, sys.stdin.readline().split())) for _ in
range(rows)]
```

```
    return Matrix(matrix_data)

def solve():
    """
    主解决函数，使用自定义的 Matrix 类和 '@' 运算符。
    """
    try:
        # 1. 读取输入并创建 Matrix 对象
        A = read_matrix_as_object()
        B = read_matrix_as_object()
        C = read_matrix_as_object()

        # 2. 执行核心计算，Python 会自动调用我们定义的 __matmul__ 和 __add__
        result_matrix = A @ B + C

        # 3. 输出结果
        for row in result_matrix.data:
            print(" ".join(map(str, row)))

    except (ValueError, IndexError):
        # ValueError: 捕获我们自己抛出的维度错误或 int() 转换错误
        # IndexError: 捕获读取数据时可能发生的错误
        print("Error!")

# 调用函数执行
solve()
```

代码运行截图（至少包含有"Accepted"）

#50261358提交状态 查看 提交 统计 提问

状态: Accepted

源代码

```
import sys

class Matrix:
    def __init__(self, data):
        """
        初始化矩阵对象。
        data: 一个二维列表 (list of lists)。
        """
```

基本信息

#: 50261358
题目: E18161
提交人: 25n2200013554
内存: 4416kB
时间: 94ms
语言: Python3
提交时间: 2025-10-08 16:24:39

E19942: 二维矩阵上的卷积运算（15分钟）

matrices, http://cs101.openjudge.cn/pctbook/E19942/

思路：使用循环嵌套，外循环确定卷积核的位置，内循环确定对应位置下卷积运算的值。

代码：

```
import sys

def solve():
    # --- 1. 读取输入 ---
```

```
    try:
        m, n, p, q = map(int, sys.stdin.readline().split())

        # 读取 m x n 的输入矩阵
        image = []
        for _ in range(m):
            image.append(list(map(int, sys.stdin.readline().split())))

        # 读取 p x q 的卷积核
        kernel = []
        for _ in range(p):
            kernel.append(list(map(int, sys.stdin.readline().split())))

    except (ValueError, IndexError):
        # 处理可能的输入格式错误
        return

    # --- 2. 确定输出维度 ---
    out_rows = m - p + 1
    out_cols = n - q + 1

    # 创建一个用于存储结果的二维列表
    result_matrix = []

    # --- 3. 执行卷积运算 ---
    # (i, j) 遍历输出矩阵的每个位置，也代表了卷积核在 image 上的左上角位置
    for i in range(out_rows):
        result_row = []
        for j in range(out_cols):

            current_sum = 0
            # (ki, kj) 遍历卷积核的每个元素
            for ki in range(p):
                for kj in range(q):
                    # 核心计算：对应位置元素相乘并累加
                    # image 的位置 = 左上角偏移(i, j) + 核内相对位置(ki, kj)
                    product = image[i + ki][j + kj] * kernel[ki][kj]
                    current_sum += product

            result_row.append(current_sum)

        result_matrix.append(result_row)

    # --- 4. 输出结果 ---
    for row in result_matrix:
        print(" ".join(map(str, row)))

# 调用函数执行
solve()
```

代码运行截图 （至少包含有"Accepted"）

#50261533提交状态 查看 提交 统计 提问

状态: Accepted

源代码

```
import sys

def solve():
    # --- 1. 读取输入 ---
    try:
        m, n, p, q = map(int, sys.stdin.readline().split())
```

基本信息

#:	50261533
题目:	E19942
提交人:	25n2200013554
内存:	3648kB
时间:	22ms
语言:	Python3
提交时间:	2025-10-08 16:33:50

M06640: 倒排索引（20分钟）

data structures, http://cs101.openjudge.cn/pctbook/M06640/

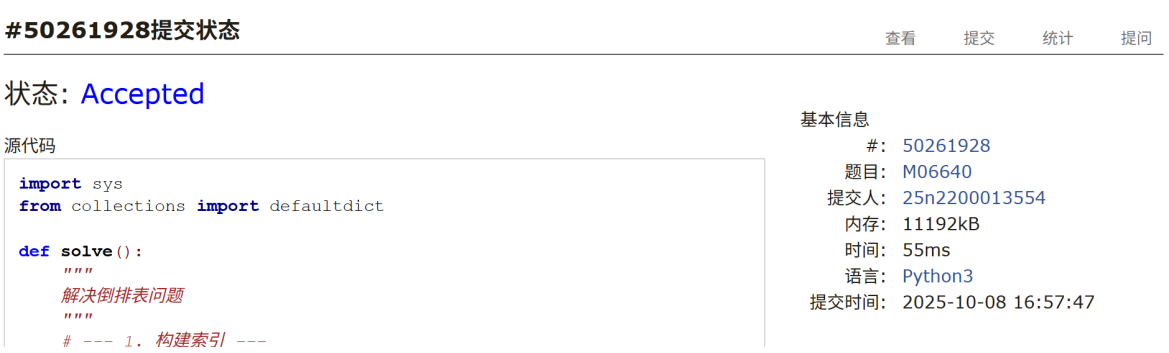
思路：为了能快速通过单词找到对应的信息，采用字典以键值对的形式存储数据，题目要求按编号从小到大排序，而集合是无序的。所以，我们需要将集合转换为列表，然后进行排序。

代码：

```
import sys
from collections import defaultdict

def solve():
    """
    解决倒排表问题
    """
    # --- 1. 构建索引 ---
    try:
        # 读取文档数 N
        num_docs = int(sys.stdin.readline())
    except (ValueError, IndexError):
        return

    # 使用 defaultdict(set)，当访问一个不存在的键时，会自动创建一个空集合
    # key: word (str), value: set of doc_ids (int)
    inverted_index = defaultdict(set)

    # 遍历 N 个文档，文档编号从 1 开始
    for doc_id in range(1, num_docs + 1):
        line_parts = sys.stdin.readline().split()

        #直接取后面的单词列表
        words_in_doc = line_parts[1:]

        for word in words_in_doc:
            # 将当前文档编号加入到对应单词的集合中
            inverted_index[word].add(doc_id)

    # --- 2. 处理查询 ---
    try:
        # 读取查询数 M
        num_queries = int(sys.stdin.readline())
    except (ValueError, IndexError):
```

```
            return

        # 遍历 M 个查询
        for _ in range(num_queries):
            # 读取查询的单词，并用 .strip() 移除末尾的换行符
            query_word = sys.stdin.readline().strip()

            # 在索引中查找单词
            if query_word in inverted_index:
                # 获取文档编号集合
                doc_ids = inverted_index[query_word]

                # 将集合转换为列表并排序
                sorted_ids = sorted(list(doc_ids))

                # 将排序后的数字列表转换为字符串并用空格连接后输出
                print(" ".join(map(str, sorted_ids)))
            else:
                # 如果单词不在索引中，输出 NOT FOUND
                print("NOT FOUND")

    # 调用函数执行
    solve()
```

代码运行截图（至少包含有"Accepted"）



E160.相交链表（45分钟）

two pinters, https://leetcode.cn/problems/intersection-of-two-linked-lists/

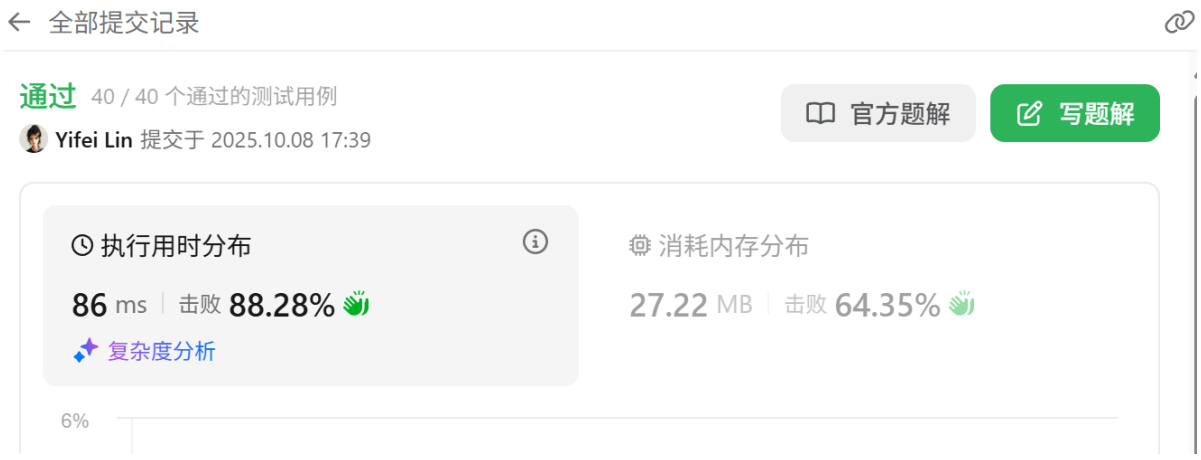
思路：进阶要求设计一个时间复杂度 O（m + n）、仅用 O（1） 内存的解决方案，采用双指针法，让两个指针分别从两个链表的头节点开始同步前进。当任何一个指针到达链表末尾时，就让它跳转到另一个链表的头节点继续前进。通过路径拼接，巧妙地让两个指针走过了相同的总距离，从而消除了两个链表原始长度的差异。当两个指针相遇的时候均前进了两段链表分开部分之和加上两个链表相交部分。

代码：

```
# Definition for singly-linked list.
class ListNode:
    def __init__(self, x):
```

```
        self.val = x
        self.next = None

class Solution:
    def getIntersectionNode(self, headA: ListNode, headB: ListNode) -> ListNode:
        # 如果任一链表为空，则不可能相交
        if not headA or not headB:
            return None

        # 初始化两个指针
        pA = headA
        pB = headB

        # 循环直到两个指针相遇
        # 如果相交，它们会在交点相遇
        # 如果不相交，它们会在都走完两个链表后同时变为 None，此时也会相遇
        while pA is not pB:
            # pA 走一步，如果到达末尾，则切换到 B 链表的头
            pA = pA.next if pA else headB

            # pB 走一步，如果到达末尾，则切换到 A 链表的头
            pB = pB.next if pB else headA

        # 循环结束时，pA (或 pB) 就是相交节点或 None
        return pA
```

代码运行截图 （至少包含有"Accepted"）



E206.反转链表（35分钟）

three pinters, recursion, https://leetcode.cn/problems/reverse-linked-list/

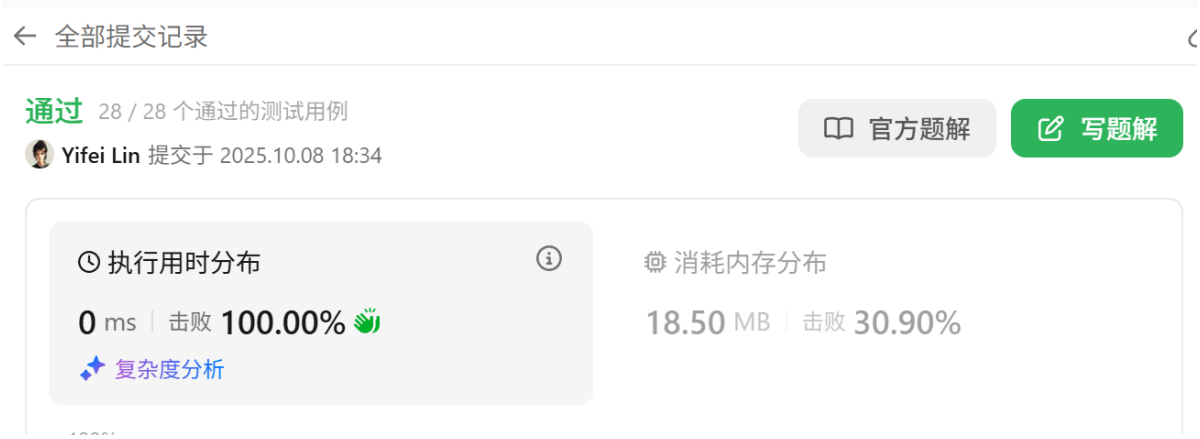
思路：用两种方法来实现。方法一，三指针法，一个是移动指针，标记正在处理的位置，第二个是反转指针，对当前节点进行反转，第三个是存储指针，用于保存下一个节点。方法二递归法，将子链表反转之后再进一步继续反转。

代码（三指针法）

```
# Definition for singly-linked list.
class ListNode:
```

```
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

class Solution:
    def reverseList(self, head: ListNode) -> ListNode:
        prev = None
        curr = head

        while curr:
            # 1. 保存下一个节点
            next_temp = curr.next
            # 2. 反转当前节点的指针
            curr.next = prev
            # 3. 移动 prev 和 curr 指针
            prev = curr
            curr = next_temp

        # 当循环结束时，prev 就是新的头节点
        return prev
```

(至少包含有"Accepted")

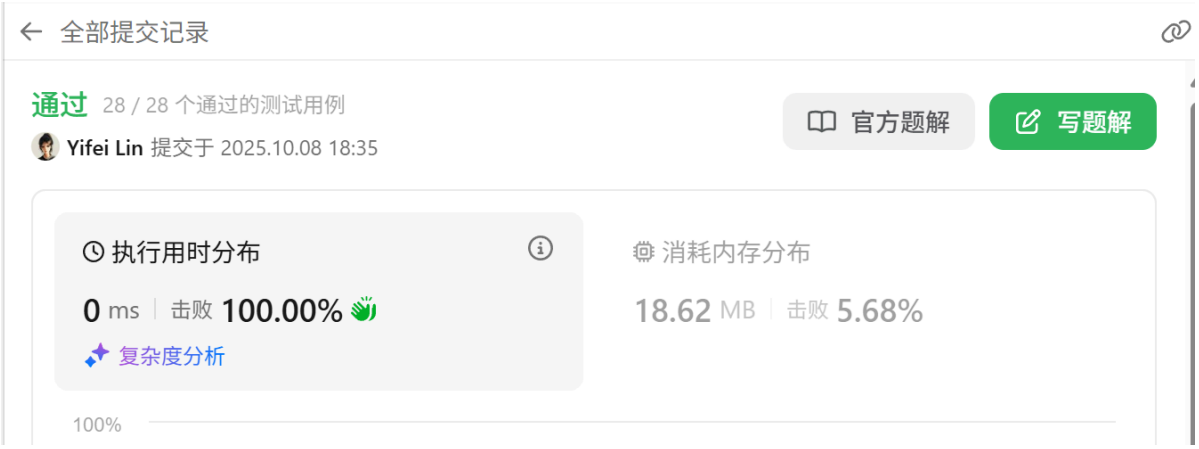


代码（递归法）

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

class Solution:
    def reverseList(self, head: ListNode) -> ListNode:
        # 1. 递归的终止条件
        if not head or not head.next:
            return head

        # 2. 递归调用，反转 head 之后的部分
        # new_head 将是反转后整个链表的头节点 (即原始链表的尾节点)
        new_head = self.reverseList(head.next)

        # 3. 反转 head 和 head.next 的指针
        # head.next 此刻是反转后子链表的尾节点
```

```
        # 我们让这个尾节点的 next 指向 head
        head.next.next = head

        # 4. 断开 head 原来的指向，防止形成环
        head.next = None

        # 5. 返回新的头节点
        return new_head
```

(至少包含有"Accepted")



T02488: A Knight's Journey（20分钟）

backtracking, http://cs101.openjudge.cn/practice/02488/

思路：采用深度优先搜索与回溯，遍历所有可能的起点，在遇到无法前进且未遍历所有格子的情况时返回上一步选择其他途径，直至找到路径或遍历所有情况后无法找到路径。

代码

```
import sys

# 将坐标 (row, col) 转换为棋盘表示法 (e.g., A1)
def to_notation(row, col):
    # chr(65) is 'A'
    return f"{chr(col + 65)}{row + 1}"

# 将棋盘表示法转换为坐标
def to_coords(notation):
    col = ord(notation[0]) - 65
    row = int(notation[1:]) - 1
    return row, col

def solve(p, q, scenario_num):
    print(f"Scenario #{scenario_num}:")

    total_squares = p * q
```

```python
    # 骑士的8个可能移动方向 (dr, dc)
    moves = [
        (-2, -1), (-2, 1), (-1, -2), (-1, 2),
        (1, -2), (1, 2), (2, -1), (2, 1)
    ]

    # --- DFS 递归函数 ---
    def dfs(path, visited):
        # 终止条件：走完了所有格子
        if len(path) == total_squares:
            return path

        # 获取当前位置
        current_row, current_col = to_coords(path[-1])

        # 生成所有可能的下一步
        next_moves = []
        for dr, dc in moves:
            next_row, next_col = current_row + dr, current_col + dc

            # 检查是否在棋盘内
            if 0 <= next_row < p and 0 <= next_col < q:
                # 检查是否已访问
                if not visited[next_row][next_col]:
                    next_moves.append(to_notation(next_row, next_col))

        # 关键：按字典序排序
        next_moves.sort()

        # 遍历排好序的移动
        for move_notation in next_moves:
            next_row, next_col = to_coords(move_notation)

            # 做出选择
            visited[next_row][next_col] = True
            path.append(move_notation)

            # 递归搜索
            result = dfs(path, visited)
            if result:  # 如果找到了解，立即返回
                return result

            # 撤销选择 (回溯)
            path.pop()
            visited[next_row][next_col] = False

        return None # 没有找到解

    # --- 主逻辑：遍历所有起点 ---
    # 按字典序生成起点
    start_points = []
    for col in range(q):
        for row in range(p):
            start_points.append(to_notation(row, col))

    for start_notation in start_points:
```

```
        start_row, start_col = to_coords(start_notation)

        # 初始化 visited 数组
        visited = [[False for _ in range(q)] for _ in range(p)]

        # 设置起点
        visited[start_row][start_col] = True
        path = [start_notation]

        # 开始搜索
        solution = dfs(path, visited)

        if solution:
            print("".join(solution))
            print()
            return

    # 如果所有起点都试过了，还是没找到解
    print("impossible")
    print()

# --- 读取输入并调用 solve ---
def main():
    try:
        num_cases = int(sys.stdin.readline())
        for i in range(1, num_cases + 1):
            p, q = map(int, sys.stdin.readline().split())
            solve(p, q, i)
    except (ValueError, IndexError):
        return

if __name__ == "__main__":
    main()
```

（至少包含有"Accepted"）

#50263606提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
import sys

# 将坐标 (row, col) 转换为棋盘表示法 (e.g., A1)
def to_notation(row, col):
    # chr(65) is 'A'
    return f"{chr(col + 65)}{row + 1}"
```

基本信息

#: 50263606
题目: 02488
提交人: 25n2200013554
内存: 3776kB
时间: 514ms
语言: Python3
提交时间: 2025-10-08 18:50:36

2. 学习总结和个人收获

链表部分对我来说有点陌生，花的时间会更多。链表反转的题目在课件中有讲到，但我对指针的运用以及递归法拆解问题的运用还不是很熟练，需要增加练习。DFS与回溯老师在课上讲解过，但是实际操作起来我感觉最后一题还是很有难度，但是使用编程方法遍历所有情况求解这种稍与数学求解分析的直觉有所出入的解法也是编程的魅力之一。