数据结构与算法B 作业4

E18161: 矩阵运算

用时：1.5 h

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

请使用`@`矩阵相乘运算符。

思路：一开始以为@运算符可以直接用，还想着现在python这么好还可以直接用@算矩阵运算，后面才发下原来要自己写个class来定义@。。由于百度上资料很有限，最后还是绕了好大的弯路，并求助AI才完成了用这个运算符算矩阵相乘。在oj上提交的时候，一直runtime error，问AI才知道在add中没有直接用列表推导式，而是每一个index都算了一遍并append进result，可能导致爆内存了。后面改用了列表推导才搞好。自己写了class才知道numpy伟大。

代码：

```
class MyMatrix:
    def __init__(self, data):
        self.data = data
        self.rows = len(data)
        self.cols = len(data[0])

    def __matmul__(self, other):
        if len(self.data[0]) != len(other.data):
            raise ValueError('Error!')

        result = [[0 for x in range(len(other.data[0]))] for x in
range(len(self.data))]
        for i in range(self.rows):
            for j in range(self.cols):
                for k in range(other.cols):
                    result[i][k] += self.data[i][j] * other.data[j][k]
        return MyMatrix(result)

    def __add__(self, other):
        if self.rows != other.rows or self.cols != other.cols:
            raise ValueError('Error!')
        result = []
        for i in range(self.rows):
            row = [self.data[i][j] + other.data[i][j] for j in
range(self.cols)]
```

```
            result.append(row)
        return MyMatrix(result)


    def __str__(self):
        return f'{self.data}'

    def __getitem__(self, index):
        return self.data[index]


class Solution:
    def matrix_cal(self, matrix):
        try:
            result = MyMatrix(matrix[0]) @ MyMatrix(matrix[1])
            result = result + MyMatrix(matrix[2])
            return [result, row[0]]
        except ValueError:
            pass
            return 'Error!'


if __name__ == '__main__':
    solution = Solution()
    matrix = [[], [], []]
    row = []
    col = []
    for i in range(3):
        temp = [int(x) for x in input().split()]
        row.append(temp[0])
        col.append(temp[1])
        for j in range(temp[0]):
            matrix[i].append([int(x) for x in input().split()])
    ans = solution.matrix_cal(matrix)
    if ans == 'Error!':
        print(ans)
    else:
        for i in range(ans[1]):
            print(' '.join([str(x) for x in ans[0][i]]))
```

#50198022提交状态

状态: Accepted

源代码

```
class MyMatrix:
    def __init__(self, data):
        self.data = data
        self.rows = len(data)
        self.cols = len(data[0])

    def __matmul__(self, other):
        if len(self.data[0]) != len(other.data):
            raise ValueError('Error!')

        result = [[0 for x in range(len(other.data[0]))] for x in range
        for i in range(self.rows):
            for j in range(self.cols):
                for k in range(other.cols):
                    result[i][k] += self.data[i][j] * other.data[j][k]
        return MyMatrix(result)

    def __add__(self, other):
        if self.rows != other.rows or self.cols != other.cols:
            raise ValueError('Error!')
        result = []
        for i in range(self.rows):
            row = [self.data[i][j] + other.data[i][j] for j in range(se
            result.append(row)
        return MyMatrix(result)
```

E19942: 二维矩阵上的卷积运算

用时：30 min

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

思路：比起要写class，这个题目就比较简单了。只要看懂题目，照着题目的意思，写一个四重嵌套的循环复现完成卷积即可。

代码：

```
class Solution:
    def convolution(self, matrices, dimensions):
        rows_core, cols_core = dimensions[1][0], dimensions[1][1]
        rows_result, cols_result = dimensions[0][0]-dimensions[1][0]+1,
```

```
dimensions[0][1]-dimensions[1][1]+1
        result = [[0 for x in range(cols_result)] for x in range(rows_result)]
        for i in range(rows_result):
            for j in range(cols_result):
                for k in range(rows_core):
                    for l in range(cols_core):
                        result[i][j] += matrices[0][i+k][j+l]*matrices[1][k]
[l]
        return result

if __name__ == '__main__':
    solution = Solution()
    matrices = []
    dimensions = []
    temp = [int(x) for x in input().split()]
    #存储矩阵和卷积核的大小
    dimensions.append([temp[0], temp[1]])
    dimensions.append([temp[2], temp[3]])
    #输入矩阵
    for rows, cols in dimensions:
        matrix = []
        for i in range(rows):
            matrix.append([int(x) for x in input().split()])
        matrices.append(matrix)
    #卷积
    ans = solution.convolution(matrices, dimensions)
    for i in range(dimensions[0][0]-dimensions[1][0]+1):
        print(' '.join([str(x) for x in ans[i]]))
```

#50198367提交状态

状态: Accepted

源代码

```
class Solution:
    def convolution(self, matrices, dimensions):
        rows_core, cols_core = dimensions[1][0], dimensions[1][1]
        rows_result, cols_result = dimensions[0][0]-dimensions[1][0]+1,
        result = [[0 for x in range(cols_result)] for x in range(rows_re
        for i in range(rows_result):
            for j in range(cols_result):
                for k in range(rows_core):
                    for l in range(cols_core):
                        result[i][j] += matrices[0][i+k][j+l]*matrices[1
        return result

if __name__ == '__main__':
    solution = Solution()
    matrices = []
    dimensions = []
    temp = [int(x) for x in input().split()]
    #存储矩阵和卷积核的大小
```

M06640: 倒排索引

用时：1 h （主要是oj上没有提供数据，出现WA的时候排查起来特别麻烦）

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

思路：读入文本和需要查找的单词，创建一个字典，将单词作为keys，每个key下建一个空列表，并遍历文本，如果文本中有需要查找的单词，就将这个单词append进key对应的列表中，最后输出即可。尤其重要的是需要考虑：同一个文本中单词可能出现多次；同一个单词可能会查询多次的情况。

代码：

```
class Solution:
    def invert(self):
        n = int(input())
        text = []
        invert_index = dict()
        for i in range(n):
            temp = input().split()
            text.append(temp)
        m = int(input())
```

```
        word_list = []
        for j in range(m):
            word_list.append(input())
            invert_index[word_list[-1]] = [0]
        for i in range(n):
            for j in range(1,int(text[i][0])+1):
                if text[i][j] in invert_index.keys():
                    if invert_index[text[i][j]][-1] != i+1:
                        invert_index[text[i][j]].append(i+1)
        return [invert_index, word_list]

if __name__ == '__main__':
    solution = Solution()
    result = solution.invert()
    for i in result[1]:
        if result[0][i] == [0]:
            print('NOT FOUND')
        else:
            print(' '.join([str(x) for x in result[0][i][1:]]))
```

状态：Accepted

源代码

```
class Solution:
    def invert(self):
        n = int(input())
        text = []
        invert_index = dict()
        for i in range(n):
            temp = input().split()
            text.append(temp)
        m = int(input())
        word_list = []
        for j in range(m):
            word_list.append(input())
            invert_index[word_list[-1]] = [0]
        for i in range(n):
            for j in range(1,int(text[i][0])+1):
                if text[i][j] in invert_index.keys():
                    if invert_index[text[i][j]][-1] != i+1:
                        invert_index[text[i][j]].append(i+1)
        return [invert_index, word_list]

if __name__ == '__main__':
```

E160.相交链表

用时：2 h （对链表不熟悉）

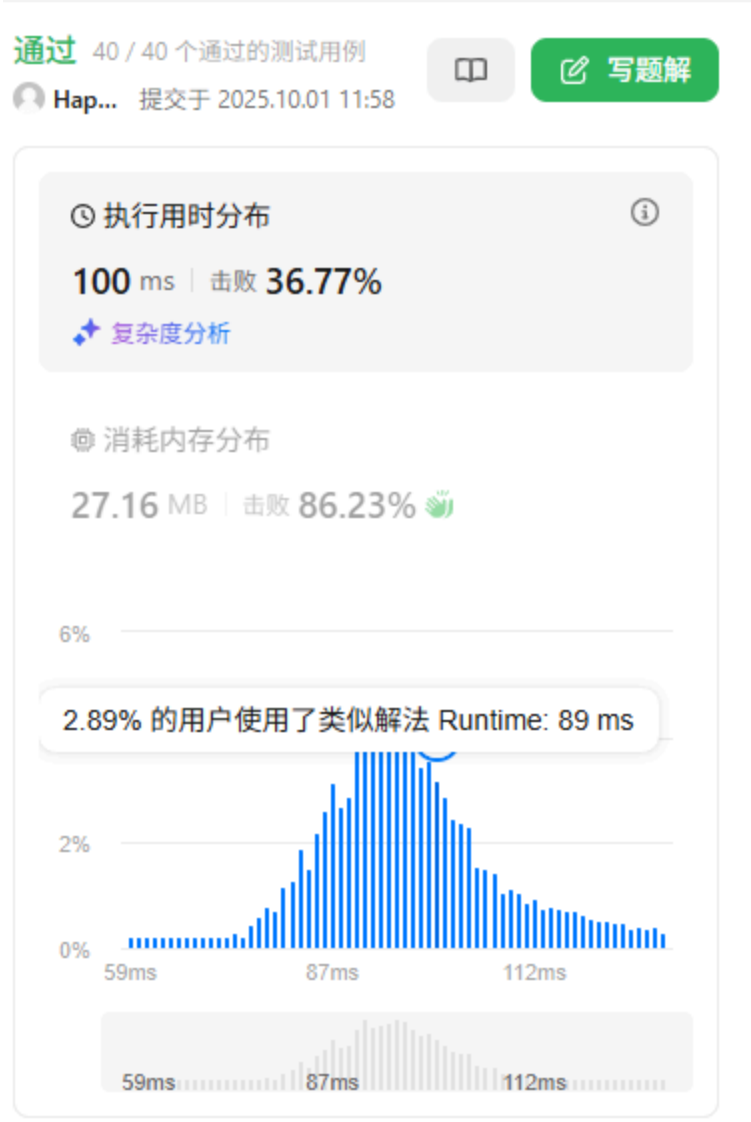
two pinters, https://leetcode.cn/problems/intersection-of-two-linked-lists/

思路：链表确实对我来说挺陌生，无论是它的表示、应用乃至题目的IO都让人有点难以理解。。后来还是找了些其他链表的题，看了题解才知道怎么回事。回到这一题，我是将两个链表的内容提取出来放到一个列表中，随后倒着遍历这个两个列表，直到出现不相同的元素，取最后一个相同的元素作为交点。需要注意两个链表完全相同的情况。

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) ->
Optional[ListNode]:
        p, q= headA, headB
        list_p = []
        list_q = []
        while p:
            list_p.append(p)
            p = p.next
        while q:
            list_q.append(q)
            q = q.next
        index_p = len(list_p)
        index_q = len(list_q)
        if list_q[index_q - 1] != list_p[index_p - 1]:
            return None
        while index_p and index_q:
            if list_q[index_q -1] != list_p[index_p - 1]:
                return list_q[index_q]
            index_p, index_q = index_p - 1, index_q - 1
        return list_q[index_q]
```



E206.反转链表

用时：2 h

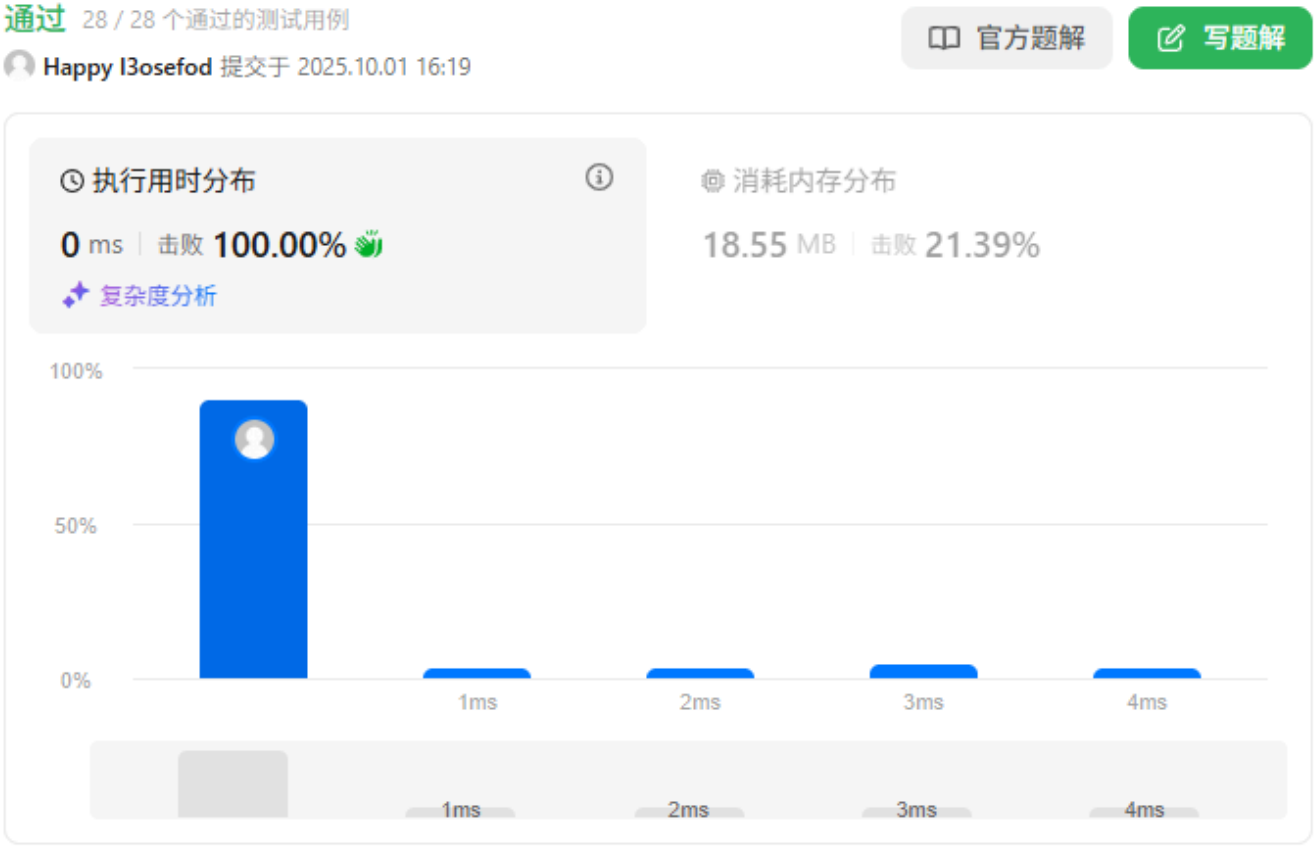
three pinters, recursion, https://leetcode.cn/problems/reverse-linked-list/

思路：这题其实不难，使用递归法提取元素，再在递归结束之后依次把上一个元素作为下一个元素的next即可。但是在写代码的时候，确实很容易绕晕过去，不知道自己要写什么、怎么写。对链表这个数据结构，我目前确实仍十分不熟悉。

代码：

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

```python
    def reverseList(self, head):
        def reverse(node, next):
            if next:
                first = reverse(next, next.next)
                next.next = node
                return first
            else:
                return node
        return(reverse(None, head))
```



代码 | Python3

```python
# 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):
```

T02488: A Knight's Journey

用时：1 h

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

思路：很经典的计概题目。望文生义，只需要照着题目的意思看棋子能否按要求遍历每一个棋盘即可。
我首先初始化骑士走的方向。注意骑士的走向需要按字典序排列以满足题目的要求，再初始化chessboard矩阵，在往上下、左右额外添加两行/列避免out of index，随后就是递归代码，将chessboard矩阵中位置定为“1”表示棋子已经走过，随后在八个走向上遍历并递归。如果某次递归后，棋子已经完全被走过（使用一个count来表示走过多少格子），说明骑士已经完全走遍棋盘。 此时，结束递归，在退出的过程中依次将走过的位置append到棋盘内，最后输出答案即可。
需要注意：如果棋子没有完全走过所有棋格即无路可走，退回上一格是需要将chessboard矩阵中对应位置恢复为“0“

代码：

```
import string


def backtrack(index_x, index_y, chessboard, count, row_x, row_y, path):
    # 标记骑士走过的位置：
    chessboard[index_x][index_y] = 1

    # count达标时，将坐标记录至path，结束递归。
    if count == row_x * row_y - 1:
        # 注意要复原棋盘
        chessboard[index_x][index_y] = 0
        path.append([index_y, index_x])
        return path

    # count未达标，则在八个方向上遍历并递归。如果某次递归返回的path不为[]，说明骑士已经完全走遍棋盘。
    # 此时，可以将当前位置append到棋盘内，结束本轮递归
    else:
        for i in range(8):
            temp_index_x = index_x + direct[i][0]
            temp_index_y = index_y + direct[i][1]
            if chessboard[temp_index_x][temp_index_y] == 0:
                backtrack(temp_index_x, temp_index_y, chessboard, count+1, row_x, row_y, path)
                if path:
                    path.append([index_y, index_x])
                    chessboard[index_x][index_y] = 0
                    return path
        chessboard[index_x][index_y] = 0


class Solution:
```

```
    def subsets(self, nums):
        all_result = []
        for i in range(nums):
            # 初始化chessboard。往上下、左右额外添加两行/列避免out of index
            chessboard_rows, chessboard_cols = [int(x) for x in
input().split()]
            chessboard = [[1 for _ in range(chessboard_cols+4)] for _ in
range(chessboard_rows+4)]
            for i in range(2, chessboard_rows+2):
                for j in range(2, chessboard_cols+2):
                    chessboard[i][j] = 0
            # 遍历矩阵作为骑士的初始位置，并开始递归
            for j in range(2, chessboard_cols + 2):
                for i in range(2, chessboard_rows+2):
                    result = backtrack(i, j, chessboard, count=0,
row_x=chessboard_rows, row_y=chessboard_cols, path=[])
                    # 这一连串break丑丑的（
                    if result:
                        all_result.append(result)
                        break
                if result:
                    break
            if result:
                continue
            else:
                all_result.append([])
        return all_result


if __name__ == '__main__':
    solution = Solution()

    # 初始化骑士走的方向。注意骑士的走向需要按字典序排列以满足题目的要求。
    direct = [[-1, -2], [1, -2], [-2, -1], [2, -1], [-2, 1], [2, 1], [-1, 2],
[1, 2]]

    # 输入参数、输出答案
    nums = int(input())
    ans = solution.subsets(nums)
    letter = list(string.ascii_uppercase)
    num_in_ans = 0
    for i in ans:
        i.reverse()
        path_str = ''
        if i:
            for j in i:
```

```
                path_str += ''.join([letter[j[0]-2], str(j[1]-1)])
        else:
            path_str += 'impossible'
        num_in_ans += 1
        print(f'Scenario #{num_in_ans}:')
        print(path_str)
        if num_in_ans != nums:
            print()
```

#50204637提交状态

状态: Accepted

源代码

```
import string


def backtrack(index_x, index_y, chessboard, count, row_x, row_y, path):
    chessboard[index_x][index_y] = 1
    if count == row_x * row_y - 1:
        chessboard[index_x][index_y] = 0
        path.append([index_y, index_x])
        return path
    else:
        for i in range(8):
            temp_index_x = index_x + direct[i][0]
            temp_index_y = index_y + direct[i][1]
            if chessboard[temp_index_x][temp_index_y] == 0:
                backtrack(temp_index_x, temp_index_y, chessboard, count
                if path:
                    path.append([index_y, index_x])
                    chessboard[index_x][index_y] = 0
                    return path
        chessboard[index_x][index_y] = 0



class Solution:
    def subsets(self, nums):
```

2. 学习总结和个人收获

其他题目都相对简单（不过由于难度上来了，每题花的时间也加了不少），但是链表那边因为完全不了解，写的十分困难。国庆期间及之后还是需要补充相关的题目作为练习。