Assignment #A: Graph starts

Updated 1830 GMT+8 Apr 22, 2025

2025 spring, Complied by 叶靖、信管

> **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

M19943:图的拉普拉斯矩阵

OOP, implementation, http://cs101.openjudge.cn/practice/19943/

要求创建Graph, Vertex两个类，建图实现。

思路：

代码：

```
def laplacian_matrix(n, m, edges):
    d = [[0] * n for _ in range(n)]
    degree = [0] * n # 度数的数组
    for a, b in edges:
        d[a][b] = 1
        d[b][a] = 1
        degree[a] += 1
        degree[b] += 1
    # 初始化拉普拉斯矩阵
```

```
        L = [[0] * n for _ in range(n)]
        for i in range(n):
            for j in range(n):
                if i == j:
                    L[i][j] = degree[i] # 主对角线元素为度数
                elif d[i][j] == 1:
                    L[i][j] = -1 # 非主对角线，若有边则 -1
        return L

    n, m = map(int, input().split())
    edges = [tuple(map(int, input().split())) for _ in range(m)]

    ans = laplacian_matrix(n, m, edges)
    for row in ans:
        print(' '.join(map(str, row)))
```

代码运行截图 （至少包含有"Accepted"）

LC78.子集

backtracking, https://leetcode.cn/problems/subsets/

思路：

代码：

```
    class Solution(object):
        def subsets(self, nums):
            res = []

            def backtrack(start, path):
                # 将当前子集加入结果中
                res.append(path)

                # 从 start 开始选择元素
                for i in range(start, len(nums)):
                    backtrack(i + 1, path + [nums[i]])

            backtrack(0, [])
            return res
```

代码运行截图 （至少包含有"Accepted"）

LC17.电话号码的字母组合

hash table, backtracking, https://leetcode.cn/problems/letter-combinations-of-a-phone-number/

思路：

代码：

```
class Solution(object):
    def letterCombinations(self, digits):
        if not digits:
            return []

        # 定义数字到字母的映射
        phone_map = {
            '2': ['a', 'b', 'c'],
            '3': ['d', 'e', 'f'],
            '4': ['g', 'h', 'i'],
            '5': ['j', 'k', 'l'],
            '6': ['m', 'n', 'o'],
            '7': ['p', 'q', 'r', 's'],
            '8': ['t', 'u', 'v'],
            '9': ['w', 'x', 'y', 'z']
        }

        # 结果存储
        res = []

        def backtrack(index, path):
            # 如果当前组合的长度等于 digits 的长度，添加到结果中
            if index == len(digits):
                res.append("".join(path))
                return

            # 获取当前数字对应的字母
            letters = phone_map[digits[index]]

            # 对每个字母进行递归
            for letter in letters:
                backtrack(index + 1, path + [letter])

        # 从第一个数字开始
        backtrack(0, [])
        return res
```

代码运行截图 （至少包含有"Accepted"）

M04089:电话号码

trie, http://cs101.openjudge.cn/practice/04089/

思路：

代码：

```
class TrieNode:
    def __init__(self):
        self.children = {}
        self.is_end_of_number = False

class Trie:
    def __init__(self):
        self.root = TrieNode()

    def insert(self, number):
        current = self.root
        for digit in number:
            if digit not in current.children:
                current.children[digit] = TrieNode()
            current = current.children[digit]
            # 如果当前节点已经是一个号码的终点，说明当前号码是前缀
            if current.is_end_of_number:
                return False
        # 如果号码已经完全插入，并且当前节点下还有其他数字存在，说明当前号码是其他号码的前缀
        if current.children:
            return False
        current.is_end_of_number = True
        return True

def check_phone_numbers(test_cases):
    results = []
    for case in test_cases:
        n = case[0]
        phone_numbers = case[1]
        trie = Trie()
        consistent = True
        for number in phone_numbers:
            if not trie.insert(number):
                consistent = False
                break
        results.append("YES" if consistent else "NO")
    return results

# 输入处理
t = int(input())  # 读入测试数据数量
test_cases = []
for _ in range(t):
    n = int(input())  # 读入电话号码的数量
    phone_numbers = [input().strip() for _ in range(n)]  # 读入电话号码
    test_cases.append((n, phone_numbers))

# 输出结果
results = check_phone_numbers(test_cases)
```

```
    for result in results:
        print(result)
```

代码运行截图 （至少包含有"Accepted"）

T28046:词梯

bfs, http://cs101.openjudge.cn/practice/28046/

思路：

代码：

```
from collections import deque, defaultdict

def bfs(start, end, word_list):
    # 词典用于存储每个中间状态对应的单词列表
    all_combo_dict = defaultdict(list)

    # 将所有单词的每个字符位置替换为 *，并记录对应的单词
    for word in word_list:
        for i in range(4):
            new_word = word[:i] + '*' + word[i+1:]
            all_combo_dict[new_word].append(word)

    # BFS 搜索
    queue = deque([start])
    prev = {start: None}  # 记录每个单词的前驱

    while queue:
        current_word = queue.popleft()

        # 如果找到了目标单词
        if current_word == end:
            path = []
            while current_word is not None:
                path.append(current_word)
                current_word = prev[current_word]
            return ' '.join(path[::-1])  # 反向返回路径

        # 对当前单词的每个可能的中间状态进行扩展
        for i in range(4):
            new_word = current_word[:i] + '*' + current_word[i+1:]
            for neighbor in all_combo_dict[new_word]:
                if neighbor not in prev:  # 如果该单词没有被访问过
                    prev[neighbor] = current_word
                    queue.append(neighbor)
```

```
        return "NO"

    # 输入处理
    n = int(input())  # 读入单词的数量
    word_list = [input().strip() for _ in range(n)]  # 读取所有单词
    start, end = input().strip().split()  # 读取起始单词和结束单词

    # 调用 BFS 来找到最短路径
    result = bfs(start, end, word_list)
    print(result)
```

代码运行截图 （至少包含有"Accepted"）

T51.N皇后

backtracking, https://leetcode.cn/problems/n-queens/

思路：

代码：

```
class Solution(object):
    def solveNQueens(self, n):
        def backtrack(row, cols, diag1, diag2, current_board):
            # 如果已经放置完所有行
            if row == n:
                result.append(["".join(row) for row in current_board])
                return

            for col in range(n):
                # 检查该列、主对角线、副对角线是否已经有皇后
                if col in cols or (row - col) in diag1 or (row + col) in diag2:
                    continue

                # 放置皇后
                current_board[row][col] = 'Q'
                cols.add(col)
                diag1.add(row - col)
                diag2.add(row + col)

                # 递归处理下一行
                backtrack(row + 1, cols, diag1, diag2, current_board)

                # 回溯，撤销放置的皇后
                current_board[row][col] = '.'
                cols.remove(col)
                diag1.remove(row - col)
                diag2.remove(row + col)
```

```
        result = []
        backtrack(0, set(), set(), set(), [['.' for _ in range(n)] for _ in range(n)])
        return result
```

代码运行截图 （至少包含有"Accepted"）

 A-06

2. 学习总结和收获

如果发现作业题目相对简单，有否寻找额外的练习题目，如“数算2025spring每日选做”、LeetCode、Codeforces、洛谷等网站上的题目。

感觉这次的作业又变得很难做了，开拓了新的题型，所以需要用到的方法也是新的。最近因为期中考，加上不少期中作业，也没时间去刷题。对自己的期末越来越没信心了，希望到时候不会挂科。