**## 1. 题目**

**### M19943:图的拉普拉斯矩阵**

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

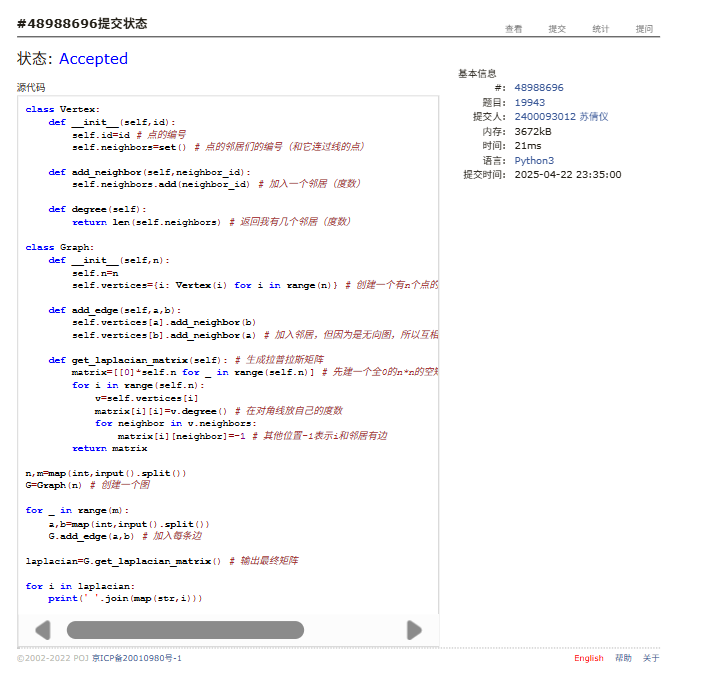
要求创建Graph, Vertex两个类，建图实现。

思路：vertex用来创建点、进行加入“邻居”和返回度数的操作，graph用来进行建图（一个从0到n-1的点编号的图，并用vertex给每个点创建一个vertex对象）、加入无向边和构造并返回拉普拉斯矩阵（创建一个n\*n，全是0的矩阵，在对角线放上度数，然后遍历该点的邻居并设为-1）的操作。最后主要调用graph中的函数来建图和输出拉普拉斯矩阵。

代码：

class Vertex:  
 def \_\_init\_\_(self,id):  
 self.id=id # 点的编号  
 self.neighbors=set() # 点的邻居们的编号（和它连过线的点）  
  
 def add\_neighbor(self,neighbor\_id):  
 self.neighbors.add(neighbor\_id) # 加入一个邻居（度数）  
  
 def degree(self):  
 return len(self.neighbors) # 返回我有几个邻居（度数）  
  
class Graph:  
 def \_\_init\_\_(self,n):  
 self.n=n  
 self.vertices={i: Vertex(i) for i in range(n)} # 创建一个有n个点的图，每个点都是一个Vertex对象  
  
 def add\_edge(self,a,b):  
 self.vertices[a].add\_neighbor(b)  
 self.vertices[b].add\_neighbor(a) # 加入邻居，但因为是无向图，所以互相做邻居  
  
 def get\_laplacian\_matrix(self): # 生成拉普拉斯矩阵  
 matrix=[[0]\*self.n for \_ in range(self.n)] # 先建一个全0的n\*n的空矩阵  
 for i in range(self.n):  
 v=self.vertices[i]  
 matrix[i][i]=v.degree() # 在对角线放自己的度数  
 for neighbor in v.neighbors:  
 matrix[i][neighbor]=-1 # 其他位置-1表示i和邻居有边  
 return matrix  
  
n,m=map(int,input().split())  
G=Graph(n) # 创建一个图  
  
for \_ in range(m):  
 a,b=map(int,input().split())  
 G.add\_edge(a,b) # 加入每条边  
  
laplacian=G.get\_laplacian\_matrix() # 输出最终矩阵  
  
for i in laplacian:  
 print(' '.join(map(str,i)))

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

  
大约用时：2小时

**### LC78.子集**

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

思路：从start（0）开始遍历，尝试将后面的数字加入path（当前子集，初始为[]），然后进行递归，每次从第i+1开始，每当到一个新的位置就将当前数字加入path，当在一个backtrack中遍历完后就跳出并回溯（pop），从新的一轮数字开始，直到循环结束

代码：

class Solution:

    def subsets(self, nums: List[int]) -> List[List[int]]:

        res=[]

        def backtrack(start,path):

            res.append(path[:])

            for i in range(start,len(nums)):

                path.append(nums[i])

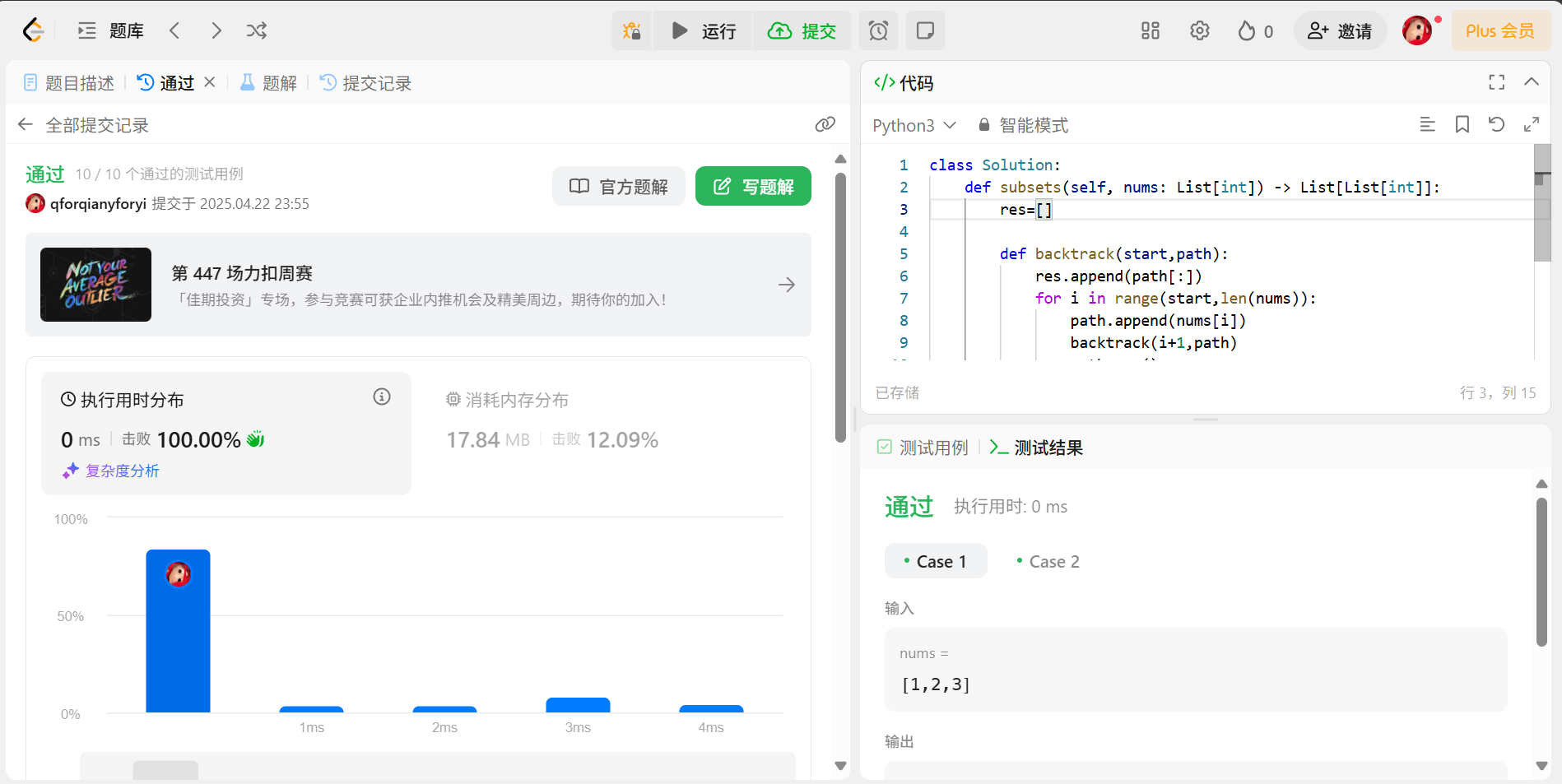
                backtrack(i+1,path)

                path.pop()

        backtrack(0,[])

        return res

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

  
大约用时：45分钟

**### LC17.电话号码的字母组合**

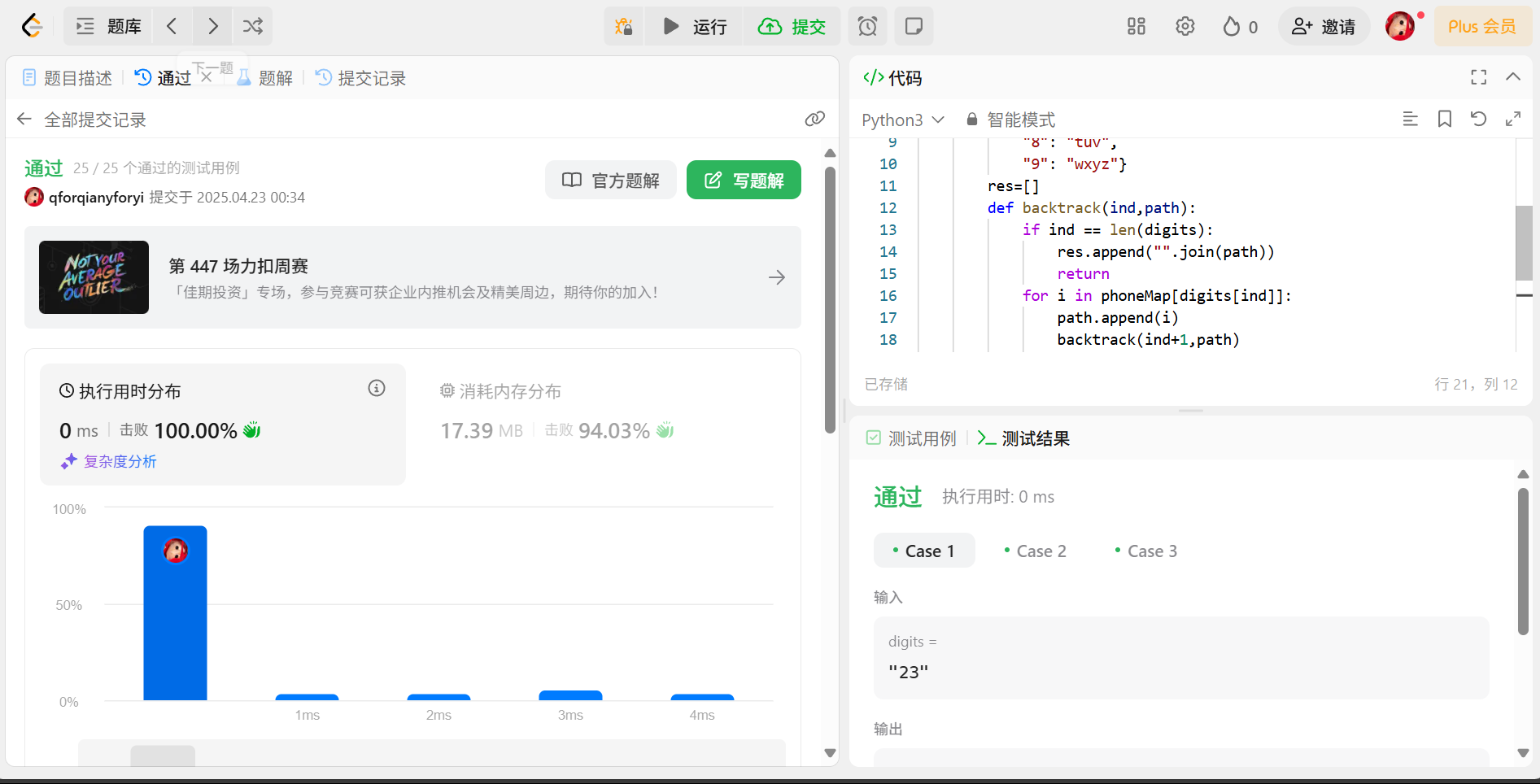
hash table, backtracking, https://leetcode.cn/problems/letter-combinations-of-a-phone-number/

思路：先创建一个字典将每个数字映射到对应的字母，然后同上一题差不多一样，从digits的第一个数字开始，找到它对应的字母集合，然后对每个字母，试着将它加入当前的path，然后继续递归处理下一个数字，当递归到达一个字母组合后进行回溯，即移除path中的最后一个字母（pop()），然后继续尝试下一个字母，直到ind（当前索引）等于digits的长度（当前路径已经是一个完整的字母组合）就将它加入res列表中。

代码：

phoneMap = {"2": "abc",  
 "3": "def",  
 "4": "ghi",  
 "5": "jkl",  
 "6": "mno",  
 "7": "pqrs",  
 "8": "tuv",  
 "9": "wxyz"}  
  
digits=input()  
  
res=[]  
def backtrack(ind,path):  
 if ind == len(digits):  
 res.append("".join(path))  
 for i in phone[digits[ind]]:  
 path.append(i)  
 backtrack(ind+1,path)  
 path.pop()  
  
if digits:  
 backtrack(0,[])  
print(res)

代码运行截图 <mark>（至少包含有"Accepted"）</mark>



大约用时：30分钟

**### M04089:电话号码**

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

思路：Trie：一种树形数据结构，可以高效地存储和检索字符串集合，每个节点表示一个字符，从根节点到某个节点的路径代表一个字符串的前缀。TrieNode类用于定义一个长度为10的列表，分别对应数字字符 '0' 到 '9'（.next）、以及标记当前节点是否是某个完整号码的结束位置（.is\_terminal）。然后在函数Trie中，遍历每个号码，从根节点开始逐个字符插入到Trie中，然后判断当前节点是否已标记为结束节点（如果是返回False，否则创建新的TrieNode并移动到下一个TrieNode子节点），或是当前节点还有子节点（any(node.next) 为 True），说明当前号码是已有号码的前缀（返回False）。如果以上条件都没有触法，说明是新号码，标记该号码的结束节点为True。最后如果Trie函数为True输出YES，否则输出NO。

代码：

class TrieNode:  
 def \_\_init\_\_(self):  
 self.next=[None]\*10 # 长度为10的数组，对应数字字符0–9  
 self.is\_terminal=False # 标记该节点是否是某个完整电话号码的末尾  
  
def trie(phone\_numbers):  
 root=TrieNode() # 根节点，不对应任何数字  
 for i in phone\_numbers: # 遍历每个电话号码  
 node=root  
 for ch in i:  
 idx=int(ch) # 字符索引  
 # 检测旧号码是新号码前缀（若当前节点已被标记为某旧号码的末尾，说明该旧号码正好是新号码在此位置之前的前缀）  
 if node.is\_terminal:  
 return False  
 # 若对应数字分支不存在则新建TrieNode，然后沿该分支继续  
 if not node.next[idx]:  
 node.next[idx]=TrieNode()  
 node=node.next[idx]  
 # 如果插入结束节点已有子节点，也冲突（号码之前已插入过/该终点下已有子节点，此前插入的某号码比当前号码更长，且以当前号码为前缀）  
 if node.is\_terminal or any(node.next):  
 return False  
 node.is\_terminal=True # 若以上冲突都未触发，则安全地把当前节点标记为一个完整号码的末尾  
 return True  
  
t=int(input())  
for \_ in range(t):  
 n=int(input())  
 nums=[input() for \_ in range(n)]  
 print("YES" if trie(nums) else "NO")

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

  
大约用时：3小时

**### T28046:词梯**

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

思路：先构建桶，桶中有依次将每个字母的字符换成\_的键以及其对应的词（邻居），然后构建图，初始化图中每个单词对应一个空邻居列表，接着对每个桶，将桶内所有单词两两配对（无向边），互为邻居，在它们的列表中添加对方。之后进行bfs，用队列存储当前探索到的路径，并每次检查末尾的词是不是目标词，否则就延展末尾节点的所有邻居（仅一字母之差的单词），生成新路径入队，直到找到目标词。

代码：

from collections import deque  
  
def graph(words):  
 buckets={} # 存放桶：键=通配标签，值=单词列表  
 graph={} # 键=单词，值=邻居单词列表  
  
 for word in words:  
 for i in range(len(word)):  
 bucket=word[:i] + '\_' + word[i+1:] # 对每个字母单词生成几种通配符形式（将第i个字母替换为\_），如FOOL→\_OOL,F\_OL,FO\_L,FOO\_  
 if bucket not in buckets:  
 buckets[bucket]=[]  
 buckets[bucket].append(word)  
 # print(buckets)  
 for word in words:  
 graph[word]=[]  
  
 for bucket\_words in buckets.values():  
 for i in range(len(bucket\_words)):  
 for j in range(i+1,len(bucket\_words)):  
 w1,w2=bucket\_words[i],bucket\_words[j]  
 graph[w1].append(w2)  
 graph[w2].append(w1) # 两两配对，生成无向边  
 # print(graph)  
 return graph  
  
def bfs(graph,start,end):  
 visited=set() # 防止重复遍历  
 queue=deque([[start]])  
  
 while queue:  
 path=queue.popleft()  
 word=path[-1]  
  
 if word == end:  
 return path  
  
 if word not in visited: # 若word未被访问  
 visited.add(word)  
 for neighbor in graph[word]: # 遍历所有graph[word]  
 new\_path=list(path)  
 new\_path.append(neighbor) # 将每个邻居追加到path的拷贝后再入队  
 queue.append(new\_path)  
  
 return None # 队列空且未找到目标  
  
n=int(input())  
word=[input() for \_ in range(n)]  
start,end=input().split()  
  
graph=graph(word)  
path=bfs(graph,start,end)  
  
if path:  
 print(' '.join(path))  
else:  
 print("NO")

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

  
大约用时：1小时30分钟

**### T51.N皇后**

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

思路：这里对角线有两个规则，即对于 ↗ /↖方向的格子，行号加列号/行号减列号是不变的，如果行号加列号或者行号减列号相同，那么两个皇后互相攻击；用数组d1和d2来分别标记之前放置的皇后的行号加列号以及行号减列号，如果当前位置在d1或d2中已经标记为True，代表当前位置会被之前放置的皇后攻击，不能放皇后。用递归来按行依次尝试放置皇后，若当前位置（当前行列和两个对角线）与已有的皇后冲突则跳过，若某行无法放置则回溯到上一行继续尝试，若可以放置则标记当前行列的对角线位置为True，直至所有行被成功放置皇后，则按照要求的格式输出皇后位置的图表。

代码：

class Solution:

    def solveNQueens(self, n: int) -> List[List[str]]:

        res=[] # 收集所有满足条件的解

        queens=[0]\*n # 长度为n的列表，用queens[r]=c 表示第r行皇后放在第c列

        column=[False]\*n # 记录列是否被占用，用col[c]=True表示第c列已有皇后，防止同列冲突

        d1=[False]\*(n\*2-1) # 记录左上到右下的对角线是否被占用，其中r+c相同的格子表示共用此对角线

        d2=[False]\*(n\*2-1) # 记录右上到左下的对角线是否被占用，其中r-c+(n-1)相同的格子表示共用此对角线

        def dfs(row): # 当前要放第r行的皇后

            if row == n: # 当所有行成功放置皇后

                res.append(['.'\*col+'Q'+'.'\*(n-1-col) for col in queens]) # 遍历queens中每行的列号col，生成输出要求的字符串（例如：c=1,n=4→.Q..）

                return

            for col,occ in enumerate(column): # 枚举第row行所有列，occ即col[col]，检查该列是否被占

                if not occ and not d1[row+col] and not d2[row-col]:  # 判断能否放皇后（三者均False才可放皇后，确保同列和两对角都无冲突）

                    queens[row]=col  # 记录第row行皇后放置在第col列

                    column[col]=d1[row+col]=d2[row-col]=True  # 标记皇后占用了第col列和两条对角线

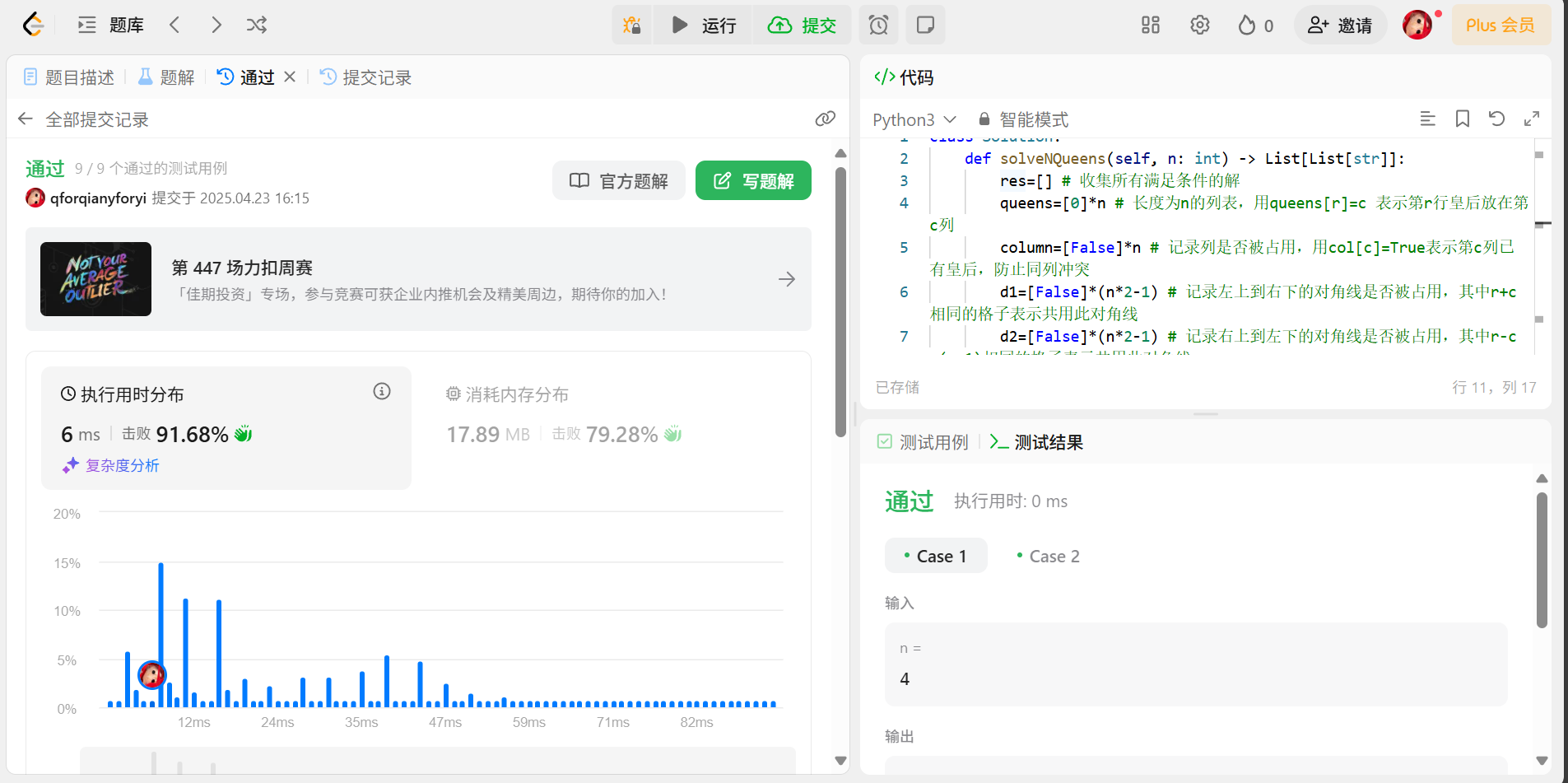
                    dfs(row+1) # 放下一行皇后

                    column[col]=d1[row+col]=d2[row-col]=False  # 回溯，撤销对列和对角线的占用标记，为尝试其他列或回退上一行做准备

        dfs(0) # 从第一（0）行开始

        return res

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

  
大约用时：2小时

**## 2. 学习总结和收获**

<mark>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。</mark>

这次的作业好难好难好难好难TT。第一题又没注意看要求...做了个矩阵的方式，最近刚学数据分析，对图还是一知半解，但是只需要调networkx就可以，手搓真的好痛苦，听了课还是搞不懂TT，最后又问了AI才稍微明白一点原理…。第二三题差不多一样，都挺好理解的（但是写起来很容易会让我的思路打结，还需要再练练）。第四题给我搞懵了，之前没接触过Trie，如果不用Trie还可以做出来，但是上网查了竟然又要类定义，虽然之后大概看懂原理了但是还是好难，真的写不出来TT。第五题可以理解但是写了几次错的代码后决定问AI了，还是对图有点陌生，所以也没有需要添加无向边这样的概念，用\_来分辨每个单词的方式也很巧妙，如果掌握了应该是很好用的技巧（如果能掌握的话…）。第六题把前三个题解的方式看懂了一遍，最后按照题解写了一遍，其中一个题解中用O(1) （额外用两个代表对角线的数组）判断当前位置会不会被之前放置的某个皇后攻击到真的很神奇！虽然以我的脑子写不出来，但是学到这个方法也让我很有成就感了hhh