Assignment #A: Graph starts

Updated 1830 GMT+8 Apr 22, 2025

2025 spring, Complied by <mark>汤伟杰，信息管理系</mark>

> **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：

感觉直接搞两个二维矩阵来操作很方便，建类有一点奇怪的感觉。点保存的是自己的值和相邻的点的列表（答案用的字典保存相邻的点和权值）；图保存的是所有点类对象的列表。然后建最后的图的过程感觉有点数学技巧了，不是很直观。。？怪怪的感觉。

代码：

```
class Vertex:
    def __init__(self, key:int):
        self.key = key
        self.neighbers = []

    def add_neighber(self, other):
        self.neighbers.append(other)

class Graph:
    def __init__(self):
        self.vertices = {}
```

```
    def add_edge(self, src, dest):
        if src not in self.vertices:
            Vsrc = Vertex(src)
            self.vertices[src] = Vsrc
        if dest not in self.vertices:
            Vdest = Vertex(dest)
            self.vertices[dest] = Vdest
        self.vertices[src].add_neighber(self.vertices[dest])

def construct(edges):
    g = Graph()
    for edge in edges:
        src, dest = edge
        g.add_edge(src, dest)
        g.add_edge(dest, src)

    ans = []
    for vertex in range(n):
        row = [0]*n
        if vertex in g.vertices:

            row[vertex] = len(g.vertices[vertex].neighbers)
            for neighber in g.vertices[vertex].neighbers:
                row[neighber.key] = -1
        ans.append(row)

    return ans


n,m=map(int,input().split())
edges=[]
for _ in range(m):
    a,b=map(int,input().split())
    edges.append((a,b))

ans=construct(edges)

for i in ans:
    print(*i)
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

#48991660提交状态

查看 提交 统计 提问

状态：Accepted

源代码

```
class Vertex:
    def __init__(self, key:int):
        self.key = key
        self.neighbers = []

    def add_neighber(self, other):
        self.neighbers.append(other)

class Graph:
    def __init__(self):
        self.vertices = {}

    def add_edge(self, src, dest):
        if src not in self.vertices:
            Vsrc = Vertex(src)
            self.vertices[src] = Vsrc
        if dest not in self.vertices:
            Vdest = Vertex(dest)
            self.vertices[dest] = Vdest
        self.vertices[src].add_neighber(self.vertices[dest])

def construct(edges):
```

基本信息

#：48991660
题目：19943
提交人：24n2400016635
内存：3668kB
时间：20ms
语言：Python3
提交时间：2025-04-23 14:20:34

LC78.子集

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

思路：

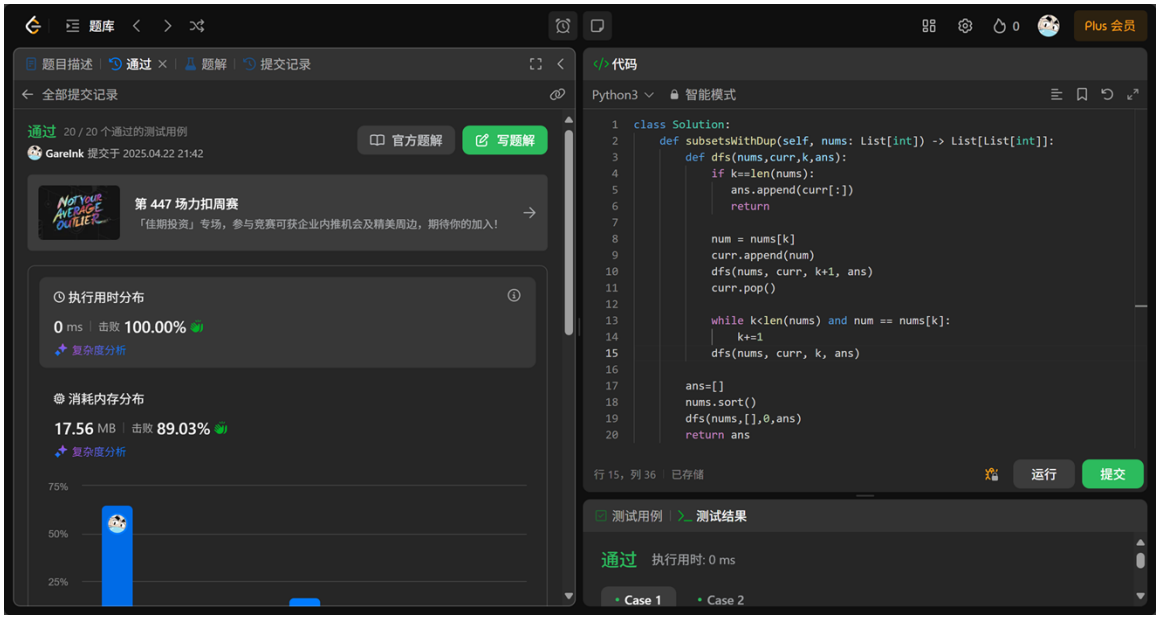
参考了灵神的题解对代码进行了优化。这道题的思路和寒假 pre 中的 01321 棋盘问题是一样的，与八皇后有一定差别。八皇后考虑的是每一个数字都要选，而这道题的每一个数字则有两种选择：选或者不选。那么普通的 dfs 一次对应的就是“选择该元素”，然后在 dfs 结束后并把这个元素 pop 掉，紧接着对下一个元素进行第二次 dfs，对应的就是“不选该元素”。然后考虑到原数组可能存在重复值，使用 while 循环将索引 k 不断右移到新元素进行第二次 dfs（题目是 90-子集 II）

代码：

```
class Solution:
    def subsetsWithDup(self, nums: List[int]) -> List[List[int]]:
        def dfs(nums,curr,k,ans):
            if k==len(nums):
               ans.append(curr[:])
               return

            num = nums[k]
            curr.append(num)
            dfs(nums, curr, k+1, ans)
            curr.pop()

            while k<len(nums) and num == nums[k]:
                k+=1
            dfs(nums, curr, k, ans)

        ans=[]
        nums.sort()
```

```
        dfs(nums,[],0,ans)
        return ans
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>



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：

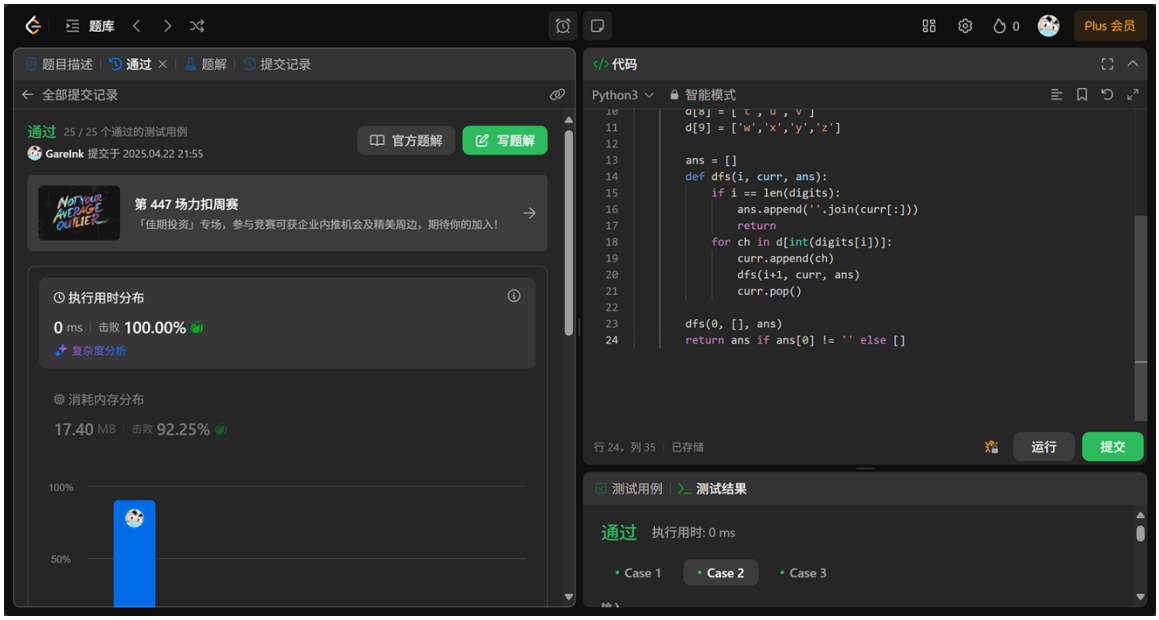
先打表，然后对 digits 的每一个数字对应的各个字母进行 dfs，是基础版 dfs。

代码：

```
class Solution:
    def letterCombinations(self, digits: str) -> List[str]:
        d = defaultdict(list)
        j = 0
        for i in range(2,7):
            for _ in range(3):
                d[i].append(chr(ord('a') + j))
                j += 1
        d[7] = ['p','q','r','s']
        d[8] = ['t','u','v']
        d[9] = ['w','x','y','z']

        ans = []
        def dfs(i, curr, ans):
            if i == len(digits):
                ans.append(''.join(curr[:]))
                return
            for ch in d[int(digits[i])]:
                curr.append(ch)
```

```
                dfs(i+1, curr, ans)
                curr.pop()

        dfs(0, [], ans)
        return ans if ans[0] != '' else []
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>



M04089:电话号码

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

思路：

先跟着 gpt 老师学了字典树，然后在模板上修改了一下就可以得到代码。考虑将号码排序，那么可能出现前缀的情况只能是：先前短的号码出现在之后长的号码中，这样就可以在插入号码的过程中进行判断，维护 `Trie` 类的 `can_build` 布尔值。对每一个节点设置 `is_end` 的布尔值，表示该字母是不是当前号码的最后一个字母。在每次插入新单词时，遍历每一个数字都检查一下当前这个数字的 `is_end` 是不是真，如果是真说明之前已经有一个短号码到此为止了，也就是存在这个前缀，就将 `Trie` 的 `can_build` 更新为 `false`。最后看的就是这个布尔值是否为真。

代码：

```
class Node:
    def __init__(self):
        self.children = dict()
        self.is_end = False

class Trie:
    def __init__(self):
        self.root = Node()
        self.can_build = True
```

```
    def insert(self, word):
        cur = self.root
        for ch in word:
            if ch not in cur.children:
                cur.children[ch] = Node()
            cur = cur.children[ch]
            # check
            if cur.is_end is True:
                self.can_build = False
        cur.is_end = True

for _ in range(int(input())):
    n = int(input())
    nums = [input() for _ in range(n)]
    nums.sort()
    trie = Trie()
    for num in nums:
        trie.insert(num)
    print(['NO','YES'][trie.can_build])
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

T28046:词梯

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

思路：

在没有读课件自己做的时候喜提 TLE，在 gpt 和课件的帮助下对两处进行了优化：

1. 在建图的时候，我刚开始使用的是 O(N^2)的两两对比，答案采用的是桶的方法：对于每个单词，把各个位置的字母模糊化，作为桶这个字典的一个键；然后将该原本单词放入这个键对应的值中。这样，每个单词都会形成 4 个键，也会放入这 4

个桶中。对每个单词这样操作之后，同一个桶里面的单词就是“只相差一个字母”的单词，对这些单词进行两两连接就好了。

2. 在 bfs 的过程中，我刚开始用的是 deque 同时存 节点和当前的总路径，目的是能够不断地将当前的总路径保存下来，遇到 end 时直接输出即可。但是这样内存会很大，学习了 gpt 的思路之后，选择用 prev 的字典来保存反向路径：`prev[next] = curr` 即下一位位置的单词存的是上一个位置的单词，这样在到达终点之后可以不断地反向回到起点。真的妙啊！

附带了 cpp 代码，写了将近 100 多行，熟悉了一些用法，感觉 python 赢在不需要变量声明，比如 `defaultdict(list)` 需要写成 `unordered_map<string, vector<string>>`，写 c 代码有一小部分时间都是在想这个变量到底是啥类型。。不过 auto 也很香（哈哈）

代码：

```
from collections import deque,defaultdict

def build(words):
    buckets = defaultdict(list)
    for word in words:
        for i in range(4):
            curr = word[:i] + '*' + word[i + 1:]
            buckets[curr].append(word)

    g = defaultdict(list)
    for bucket in buckets.values():
        for i in range(len(bucket)):
            for j in range(i + 1, len(bucket)):
                a,b = bucket[i], bucket[j]
                g[a].append(b)
                g[b].append(a)

    return g

def bfs(g, start, end):
    q = deque([start])
    prev = {start:None}
    found = False

    while q:
        if found:
            break
        for _ in range(len(q)):
            curr = q.popleft()
            if curr == end:
                found = True
                break
            for next in g[curr]:
                if next not in prev:
                    q.append(next)
                    prev[next] = curr

    if not found:
        return 'NO'
```

```
    else:
        path=[]
        cur=end
        while cur:
            path.append(cur)
            cur = prev[cur]
        return ' '.join(path[::-1])
n=int(input())
words=[input() for _ in range(n)]
start, end = input().split()
g = build(words)
print(bfs(g, start, end))
```

cpp:

```
#include <bits/stdc++.h>
using namespace std;

using dv = unordered_map< string, vector<string> >;
using ds = unordered_map< string, string >;

dv build(const vector<string>& words) {
    dv buckets;
    for (auto word : words) {
        for (int i = 0; i < 4; i++) {
            string curr = word.substr(0, i) + '*' + word.substr(i + 1);
            buckets[curr].push_back(word);
        }
    }

    dv g;
    for (auto [_, bucket] : buckets) {
        for (int i = 0; i < bucket.size(); i++) {
            for (int j = i + 1; j < bucket.size(); j++) {
                string a = bucket[i], b = bucket[j];
                g[a].push_back(b);
                g[b].push_back(a);
            }
        }
    }

    return g;
}

void bfs(dv g, const string& start, const string& end) {
    deque<string> q;
    q.push_back(start);

    ds prev;
    prev[start] = "";
    bool found = false;

    while (q.size()) {
        if (found) {
            break;
```

```
        }
        for (int i = 0; i < q.size(); i++) {
            string curr = q.front();
            q.pop_front();
            if (curr == end) {
                found = true;
                break;
            }
            for (string next : g[curr]) {
                if (prev.find(next) == prev.end()) {
                    prev[next] = curr;
                    q.push_back(next);
                }
            }
        }
    }

    if (!found) {
        cout << "NO" << endl;
    } else {
        vector<string> path;
        string curr = end;
        while (curr != "") {
            path.push_back(curr);
            curr = prev[curr];
        }
        reverse(path.begin(), path.end());
        string ans;
        for (int i = 0; i < path.size(); i++) {
            if (i == 0) {
                ans += path[i];
            } else {
                ans += " " + path[i];
            }
        }
        cout << ans << endl;
    }
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);

    int t;
    cin >> t;

    vector<string> words;
    for (int i = 0; i < t; i++) {
        string w;
        cin >> w;
        words.push_back(w);
    }

    string start, end;
    cin >> start >> end;
```

```
    dv g = build(words);
    bfs(g, start, end);
    return 0;
}
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

T51.N 皇后

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

思路：

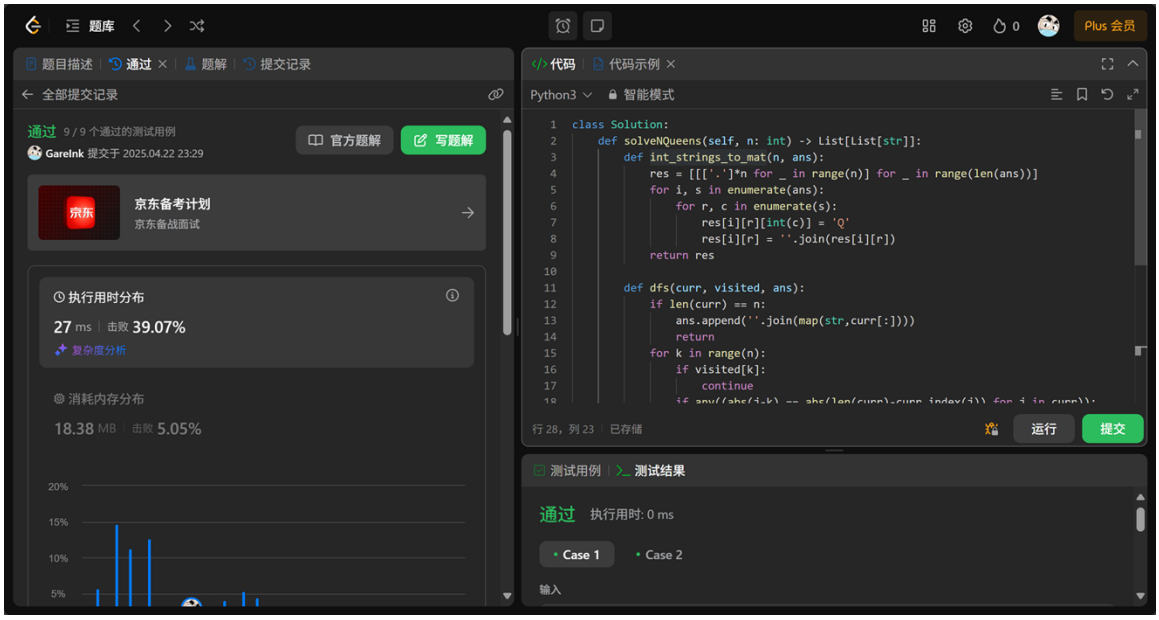
借用了以前做八皇后的思路，先用 dfs 把可能的结果用一串列索引数字保存起来，然后将这些数字转为棋盘的字符串形式。最后这个转换过程 debug 半天，都是 python 字符串不支持修改的原因。。。

代码：

```
class Solution:
    def solveNQueens(self, n: int) -> List[List[str]]:
        def int_strings_to_mat(n, ans):
            res = [[['.']*n for _ in range(n)] for _ in range(len(ans))]
            for i, s in enumerate(ans):
                for r, c in enumerate(s):
                    res[i][r][int(c)] = 'Q'
                    res[i][r] = ''.join(res[i][r])
            return res

        def dfs(curr, visited, ans):
            if len(curr) == n:
                ans.append(''.join(map(str,curr[:])))
                return
```

```
            for k in range(n):
                if visited[k]:
                    continue
                if any((abs(j-k) == abs(len(curr)-curr.index(j)) for j in
curr)):
                    continue
                curr.append(k)
                visited[k] = 1
                dfs(curr, visited, ans)
                curr.pop()
                visited[k] = 0

        ans=[]
        dfs([],[0]*n,ans)
        return int_strings_to_mat(n,ans)
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>



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>

要抓紧学课件了，虽然 bfs 和 dfs 的题目都是计概的题，但是后面一些新的知识点要提前熟悉！本周学了 KMP 算法，那个 next 数组花费了一个晚上才理解，不过做题的时候把模板写上去就好了，效率很高。其实也额外学了 Trie，没想到恰好作业里面出现了，真巧！

目前学了 cpp 的 STL，用这门语言做题有了一定的手感了，到时候 python 超时了就用 cpp 码一遍试试（