Assignment #8: 图论：概念、遍历，及 树算

Updated 1919 GMT+8 Apr 8, 2024

2024 spring, Complied by 李佳霖，心理与认知科学学院

说明：

1）请把每个题目解题思路（可选），源码Python, 或者C++（已经在Codeforces/Openjudge上AC），截图（包含Accepted），填写到下面作业模版中（推荐使用 typora https://typoraio.cn ，或者用word）。AC 或者没有AC，都请标上每个题目大致花费时间。

2）提交时候先提交pdf文件，再把md或者doc文件上传到右侧“作业评论”。Canvas需要有同学清晰头像、提交文件有pdf、"作业评论"区有上传的md或者doc附件。

3）如果不能在截止前提交作业，请写明原因。

编程环境

（请改为同学的操作系统、编程环境等）

操作系统：macOS Sonoma 14.0

Python编程环境：VSCode

C/C++编程环境：Mac terminal vi (version 9.0.1424), g++/gcc (Apple clang version 14.0.3, clang-1403.0.22.14.1)

1. 题目

19943: 图的拉普拉斯矩阵

matrices, http://cs101.openjudge.cn/practice/19943/

请定义Vertex类，Graph类，然后实现

思路：参考教材和答案进行实现

代码

```
#
class Vertex:
```

```
    def __init__(self, key):
        self.id = key
        self.connectedTo = {} #记录与其相连的顶点，以及每一条边的权重

    # 添加从一个顶点到另一个的连接
    def addNeighbor(self, nbr, weight = 0):
        self.connectedTo[nbr] = weight

    def __str__(self):
        return str(self.id) + ' connectedTo: ' + str([x.id for x in self.connectedTo])

    # 返回邻接表中的所有顶点
    def getConnections(self):
        return self.connectedTo.keys()

    def getId(self):
        return self.id

    # 返回从当前顶点到以参数传入的顶点之间的边的权重
    def getWeight(self, nbr):
        return self.connectedTo[nbr]

class Graph:
    def __init__(self):
        self.vertList = {}
        self.numVertices = 0

    def addVertex(self, key):
        self.numVertices = self.numVertices + 1
        newVertex = Vertex(key)
        self.vertList[key] = newVertex
        return newVertex

    def getVertex(self, n):
        if n in self.vertList:
            return self.vertList[n]
        else:
            return None

    def __contains__(self, n):
        return n in self.vertList

    def addEdge(self, f, t, cost = 0):
        if f not in self.vertList:
            nv = self.addVertex(f)
        if t not in self.vertList:
            nv = self.addVertex(t)
```

```
        self.vertList[f].addNeighbor(self.vertList[t], cost)

    def getVertices(self):
        return self.vertList.keys()

    def __iter__(self):
        return iter(self.vertList.values())

n, m = map(int, input().split())
g = Graph()

for i in range(n):
    g.addVertex(i)

for i in range(m):
    u, v= map(int, input().split())
    g.addEdge(u, v)
    g.addEdge(v, u)

L = []
for vertex in g:
    row = [0] * n
    row[vertex.getId()] = len(vertex.getConnections())
    for neighbor in vertex.getConnections():
        row[neighbor.getId()] = -1
    L.append(row)

for row in L:
    print(' '.join(map(str, row)))
```

代码运行截图 （至少包含有"Accepted"）

#44630523提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
class Vertex:
    def __init__(self, key):
        self.id = key
        self.connectedTo = {} #记录与其相连的顶点，以及每一条边的权重

    # 添加从一个顶点到另一个的连接
    def addNeighbor(self, nbr, weight = 0):
        self.connectedTo[nbr] = weight
```

基本信息

#: 44630523
题目: 19943
提交人: 李佳霖2000013713
内存: 3736kB
时间: 29ms
语言: Python3
提交时间: 2024-04-13 14:45:49

18160: 最大连通域面积

matrix/dfs similar, http://cs101.openjudge.cn/practice/18160

思路：很典型的深搜题目

代码

```
#
def dfs(grid, row, col):
    if row < 0 or col < 0 or row >= len(grid) or col >= len(grid[0]) or grid[row][col] != 'W':
        return 0

    # 将当前位置标记为已访问
    grid[row][col] = '.'
    size = 1

    # 遍历当前位置的周围八个位置
    for dr in [-1, 0, 1]:
        for dc in [-1, 0, 1]:
            size += dfs(grid, row + dr, col + dc)

    return size

T = int(input())
for _ in range(T):
    N, M = map(int, input().split())
    grid = [list(input()) for _ in range(N)]

    max_area = 0
    # 遍历棋盘的每个位置
    for i in range(N):
        for j in range(M):
            if grid[i][j] == 'W':
                area = dfs(grid, i, j)
                max_area = max(max_area, area)

    print(max_area)
```

代码运行截图 （至少包含有"Accepted"）

#44630913提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
# 定义DFS函数
def dfs(grid, row, col):
    if row < 0 or col < 0 or row >= len(grid) or col >= len(grid[0]) or
        return 0

    # 将当前位置标记为已访问
    grid[row][col] = '.'
    size = 1
```

基本信息

#: 44630913
题目: 18160
提交人: 李佳霖2000013713
内存: 3744kB
时间: 124ms
语言: Python3
提交时间: 2024-04-13 14:58:18

sy383: 最大权值连通块

https://sunnywhy.com/sfbj/10/3/383

思路：类似第一题，对每个节点加入一个权重属性，然后使用dfs来递归地遍历节点，累加所有的权重和，并使用max求出最大的

代码

```
#
class Vertex:
  def __init__(self, key, weight):
    self.id = key
    self.weight = weight
    self.connectedTo = {} #记录与其相连的顶点，以及每一条边的权重

  # 添加从一个顶点到另一个的连接
  def addNeighbor(self, nbr, weight = 0):
    self.connectedTo[nbr] = weight

  def __str__(self):
    return str(self.id) + ' connectedTo: ' + str([x.id for x in self.connectedTo])

  # 返回邻接表中的所有顶点
  def getConnections(self):
    return self.connectedTo.keys()

  def getId(self):
    return self.id

  # 返回从当前顶点到以参数传入的顶点之间的边的权重
```

```
    def getWeight(self, nbr):
        return self.connectedTo[nbr]

class Graph:
    def __init__(self):
        self.vertList = {}
        self.numVertices = 0

    def addVertex(self, key, weight):
        self.numVertices = self.numVertices + 1
        newVertex = Vertex(key, weight)
        self.vertList[key] = newVertex
        return newVertex

    def getVertex(self, n):
        if n in self.vertList:
            return self.vertList[n]
        else:
            return None

    def __contains__(self, n):
        return n in self.vertList

    def addEdge(self, f, t, cost = 0):
        if f not in self.vertList:
            nv = self.addVertex(f)
        if t not in self.vertList:
            nv = self.addVertex(t)
        self.vertList[f].addNeighbor(self.vertList[t], cost)

    def getVertices(self):
        return self.vertList.keys()

    def __iter__(self):
        return iter(self.vertList.values())

def dfsTravel(G):
    def dfs(v):
        sumw = G[v].weight  # 初始化当前连通块的权值
        visited[v] = True
        for u in G[v].connectedTo:
            if not visited[u.getId()]:
                sumw += dfs(u.getId())
        return sumw

    n = len(G)
    visited = [False for i in range(n)]
```

```
    max_weight = 0
    for i in range(n):
        if not visited[i]:
            new_weight = dfs(i)
            max_weight = max(max_weight, new_weight)

    return max_weight

n, m = map(int, input().split())
weight = list(int(_) for _ in input().split())
g = Graph()
for i in range(n):
    g.addVertex(i, weight[i])

for i in range(m):
    u, v= map(int, input().split())
    g.addEdge(u, v)
    g.addEdge(v, u)

print(dfsTravel(g.vertList))
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）

题目 题解

最大权值连通块

通过数 587 提交数 1093 难度 中等 显示标签

题目描述

现有一个共n个顶点、m条边的无向图（假设顶点编号为从 `0` 到 `n-1`），每个顶点有各自的权值。我们把一个连通块中所有顶点的权值之和称为这个连通块的权值。求图中所有连通块的最大权值。

输入描述

第一行两个整数n、m（$1 \le n \le 100, 0 \le m \le \frac{n(n-1)}{2}$），分别表示顶点数和边数；

第二行n个用空格隔开的正整数（每个正整数不超过 `100`），表示n个顶点的权值。

接下来m行，每行两个整数u、v（$0 \le u \le n-1, 0 \le v \le n-1, u \ne v$），表示一条边的两个端点的编号。数据保证不会有重边。

代码书写 Python

```
class Vertex:
    def __init__(self, key, weight):
        self.id = key
        self.weight = weight
        self.connectedTo = {} #记录与其相连的顶点，以及每一条边的权重

    # 添加从一个顶点到另一个的连接
    def addNeighbor(self, nbr, weight = 0):
        self.connectedTo[nbr] = weight

    def __str__(self):
        return str(self.id) + ' connectedTo: ' + str([x.id for x in se

    # 返回邻接表中的所有顶点
    def getConnections(self):
```

测试输入 提交结果 历史提交

完美通过 查看题解

100% 数据通过测试

运行时长: 0 ms

03441: 4 Values whose Sum is 0

data structure/binary search, http://cs101.openjudge.cn/practice/03441

思路：用字典存储前两个list中所有可能的结果及出现次数，计算另外两个list中两两加和的结果，如果在字典中查找是否有相反数，有则累加。

代码

```
#
def count_quadruplets(A, B, C, D):
    # 存储 a + b 的结果及其出现的次数
    ab_sum_count = {}

    # 统计所有可能的 a + b 的结果
    for a in A:
        for b in B:
            ab_sum = a + b
            ab_sum_count[ab_sum] = ab_sum_count.get(ab_sum, 0) + 1

    # 遍历 C 和 D 的组合，查找是否存在相加结果的相反数
    count = 0
    for c in C:
        for d in D:
            cd_sum = c + d
            # 如果存在相反数，累加数量
            if -cd_sum in ab_sum_count:
                count += ab_sum_count[-cd_sum]

    return count

n = int(input())
A, B, C, D = [], [], [], []
for _ in range(n):
    a, b, c, d = map(int, input().split())
    A.append(a)
    B.append(b)
    C.append(c)
    D.append(d)

result = count_quadruplets(A, B, C, D)
print(result)
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）

#44633667提交状态

查看 提交 统计 提问

状态: Accepted

基本信息

#: 44633667
题目: 03441
提交人: 李佳霖2000013713
内存: 171728kB
时间: 3107ms
语言: Python3
提交时间: 2024-04-13 16:35:45

源代码

```
def count_quadruplets(A, B, C, D):
    # 存储 a + b 的结果及其出现的次数
    ab_sum_count = {}

    # 统计所有可能的 a + b 的结果
    for a in A:
        for b in B:
            ab_sum = a + b
            ab_sum_count[ab_sum] = ab_sum_count.get(ab_sum, 0) + 1
```

04089: 电话号码

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

Trie 数据结构可能需要自学下。

思路：先学习了一下什么是Trie结构，这里能用的原因是因为它刚好关心的就是前缀，所以从根节点向下找刚好能实现这个目的。能学到的一点比如说对于具有相同一部分序列的字符串而言，比如976和911，两个具有共同元素9，再第二位开始分叉，Trie便能够解决这种问题提高查找效率

代码

```
#
class TrieNode:
  def __init__(self):
    self.children = {} # 子节点，使用字典实现

class Trie:
  def __init__(self):
    self.root = TrieNode() # 创建 Trie 的根节点

  def insert(self, word):
    node = self.root
    for char in word:
      if char not in node.children:
        node.children[char] = TrieNode() # 如果节点不存在，则创建一个新节点
      node = node.children[char] # 移动到下一个节点

  def search(self, word):
    node = self.root
```

```
        for char in word:
            if char not in node.children:
                return False # 如果字符不存在，则单词不存在
            node = node.children[char] # 移动到下一个节点
        return True

t = int(input())
for i in range(t):
    trie = Trie()
    n = int(input())
    numbers = []
    for j in range(n):
        number = input()
        numbers.append(number)
    numbers.sort(reverse= True)
    res = 0
    for number in numbers:
        res += trie.search(number)
        trie.insert(number)

    print('NO' if res > 0 else 'YES')
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）

#44640063提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
class TrieNode:
    def __init__(self):
        self.children = {}  # 子节点，使用字典实现

class Trie:
    def __init__(self):
        self.root = TrieNode()  # 创建 Trie 的根节点
```

基本信息

#: 44640063
题目: 04089
提交人: 李佳霖2000013713
内存: 24956kB
时间: 391ms
语言: Python3
提交时间: 2024-04-13 21:00:10

04082: 树的镜面映射

http://cs101.openjudge.cn/practice/04082/

思路：借助题解和ChatGPT和Python Tutor才勉强能够读懂代码

多叉树转换成伪满二叉树的转换规则如下：

1. **层次遍历转换**：从多叉树的根节点开始，按照层次遍历的顺序遍历每个节点，并逐个将其转换为二叉树节点。在转换过程中，对于每个多叉树节点，将其的每个子节点依次添加为其在二叉树中的右孩子，并将它的兄弟节点依次添加为其右孩子的右孩子，以此类推，直到将所有子节点都添加到二叉树中。
2. **增加虚拟节点**：如果多叉树的某个节点的子节点个数小于 2，那么在转换成的二叉树中，需要添加虚拟节点来保持二叉树的结构。具体地，对于每个多叉树节点，如果其子节点个数为 0，则在其转换后的二叉树节点右侧添加一个虚拟节点；如果子节点个数为 1，则在其转换后的二叉树节点右侧添加一个虚拟节点，并将其子节点添加为虚拟节点的右孩子。
3. **特殊处理根节点**：根节点的转换略有不同，它在转换后的二叉树中仍然是根节点，但其右孩子为其在多叉树中的第一个子节点，而不是其他兄弟节点。因此，在转换过程中，需要将多叉树的根节点转换为二叉树的根节点，并将其第一个子节点添加为右孩子，其余子节点按照上述规则依次添加。

缩略版：对于多叉树节点，只将左孩保留原来的关系，左孩的兄弟节点依次向下连接为右孩子节点。对于子节点数小于2的节点，补充伪节点

Build_tree函数原理：

1. 从列表的第一个元素开始，每个元素都表示一个节点的值和是否有子节点的标记。如果标记为 '0'，表示该节点有子节点，否则表示该节点没有子节点。
2. 在处理每个节点时，先创建一个树节点，并将其值设为当前元素的值。然后根据标记的情况决定是否递归构建子节点。
3. 如果当前节点有子节点，则依次递归构建子节点，并将子节点添加到当前节点的子节点列表中。
4. 返回当前节点及下一个待处理的元素的索引。

为什么if tempList[index][1] == '0':下有两遍递归：因为输入的是前序遍历序列，即自己，左子，右子的顺序，例如题目中的d节点，因为其两个子节点都没有孩子，所以是对应的数字都是1，因此d0后面跟着的就是两个子节点。如果子节点还有子节点，则会继续向下递归，到头就会返回

Print_tree函数原理：

结合题目，函数里实际上是这样的运行的：先把a添加进栈，然后把p设为其右子节点，然后其右子节点的返回值是None，所以跳出第一个while循环。第二个while循环中把a从栈里pop出来并添加进队列。进入第三个while循环，把p再设置为a，检查其左子节点（伪满二叉树中只有左子节点和其父节点才真正满足多叉树中的隶属关系），输出a。然后依次把b, c, f添加进栈（添加b的时候先检查了其左子节点，但发现是None），到f的时候跳出第三个while循环中的while循环，开始把栈中的元素反向pop出添加进队列。先输出f和c，到c的时候检查发现有左子节点，于是把p设置成d，进入里面的while循环后，先把d添加到栈中，再把p设置成其右子节点c，递归发现没有子节点后，进入下一个while循环中，先pop出e，再pop出d，添加到队列中，先输出b，再输出e，再输出d，这是p指定的是节点d，检查其左子节点是$，结束第三个循环。

代码

```
#
from collections import deque

class TreeNode:
```

```
    def __init__(self, x):
        self.x = x
        self.children = []

def create_node():
    return TreeNode('')

def build_tree(tempList, index):
    node = create_node()
    node.x = tempList[index][0]
    if tempList[index][1] == '0':
        index += 1
        child, index = build_tree(tempList, index)
        node.children.append(child)
        index += 1
        child, index = build_tree(tempList, index)
        node.children.append(child)
    return node, index

def print_tree(p):
    Q = deque()
    s = deque()

    # 遍历右子节点并将非虚节点加入栈s
    while p is not None:
        if p.x != '$':
            s.append(p)
        p = p.children[1] if len(p.children) > 1 else None

    # 将栈s中的节点逆序放入队列Q
    while s:
        Q.append(s.pop())

    # 宽度优先遍历队列Q并打印节点值
    while Q:
        p = Q.popleft()
        print(p.x, end=' ')

        # 如果节点有左子节点，将左子节点及其右子节点加入栈s
        if p.children:
            p = p.children[0]
            while p is not None:
                if p.x != '$':
                    s.append(p)
                p = p.children[1] if len(p.children) > 1 else None

            # 将栈s中的节点逆序放入队列Q
```

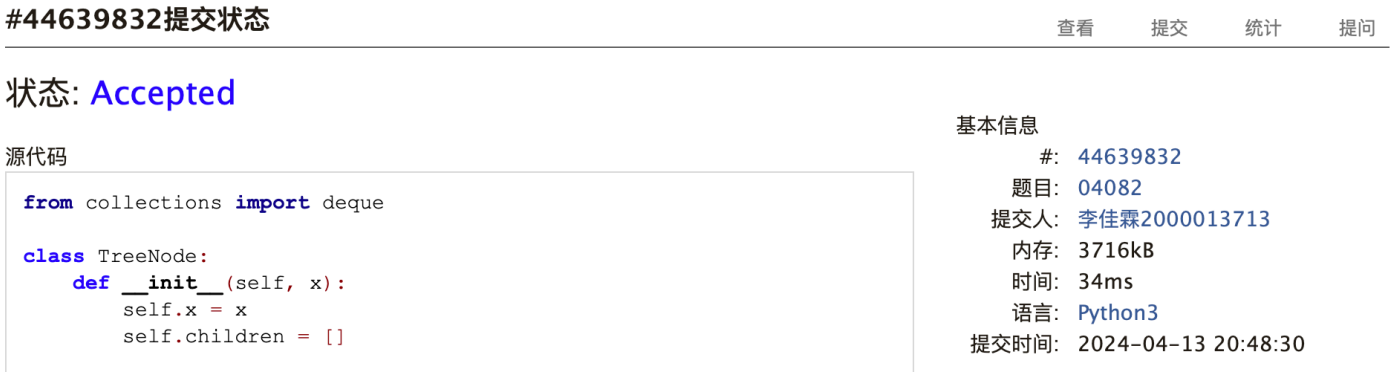
```
        while s:
            Q.append(s.pop())


n = int(input())
tempList = input().split()

# 构建多叉树
root, _ = build_tree(tempList, 0)

# 执行宽度优先遍历并打印镜像映射序列
print_tree(root)
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）



#44639832提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
from collections import deque

class TreeNode:
    def __init__(self, x):
        self.x = x
        self.children = []
```

基本信息

#:	44639832
题目:	04082
提交人:	李佳霖2000013713
内存:	3716kB
时间:	34ms
语言:	Python3
提交时间:	2024-04-13 20:48:30

2. 学习总结和收获

如果作业题目简单，有否额外练习题目，比如：OJ“2024spring每日选做”、CF、LeetCode、洛谷等网站题目。

这周题目对我来说难度还是比较大的，很多知识点都是借助答案和Python Tutor反过来理解，可能还需要一段时间的消化。