18 第六讲 作业记录（一）

笔记本: 浙江大学《数据结构》
创建时间: 2025/4/13 19:08 **更新时间:** 2025/4/13 19:25
作者: panhengye@163.com
URL: https://pintia.cn/problem-sets/1873565885118418944/exam/problems/type/7?...

06-图1 列出连通集

提交结果

题目	用户	提交时间
06-图1	飞翔的小师弟	2025/04/13 19:05:06
编译器	内存	用时
Python (python3)	3404 / 65536 KB	20 / 400 ms
状态	分数	评测时间
答案正确	25 / 25	2025/04/13 19:05:06

评测详情

测试点	提示	内存(KB)	用时(ms)	结果	得分
0	sample 两种顺序不同，也有相同，有未出现的单个顶点	3384	19	答案正确	15 / 15
1	第1个是单独点，最大N	3404	18	答案正确	8 / 8
2	N和E最小	3292	20	答案正确	2 / 2

【代码记录】

```
from collections import defaultdict, deque


def dfs(graph, start, visited):

    """
    采用深度优先方法，从给定的位置开始遍历未访问过的结点
    """

    connected_list = []
    stack = [start]
    while stack:
        vertex = stack.pop()
        if vertex not in visited:
            visited.add(vertex)
            connected_list.append(vertex)
            # 通过排序实现按编号递增的顺序访问邻接点,由于栈后进先出，所以要降序
            stack.extend(sorted(graph[vertex], reverse=True))
    return connected_list


def bfs(graph, start, visited):

    """
```

```
    采用广度优先方法，从给定的位置开始遍历未访问过的结点
    """

    connected_list = []
    queue = deque([start])
    while queue:
        vertex = queue.popleft()
        if vertex not in visited:
            visited.add(vertex)
            connected_list.append(vertex)
            # 通过排序实现按编号递增的顺序访问邻接点,由于队列先进先出，所以要升序
            queue.extend(sorted(graph[vertex]))
    return connected_list


def find_connected_sets(graph, n):

    """
    确保从编号最小的顶点出发，遍历图中所有的结点
    """

    visited_dfs = set()
    visited_bfs = set()
    dfs_list = []
    bfs_list = []
    for i in range(n):
        if i not in visited_dfs:
            dfs_connected_list = dfs(graph, i, visited_dfs)
            dfs_list.append(dfs_connected_list)
        if i not in visited_bfs:
            bfs_connected_list = bfs(graph, i, visited_bfs)
            bfs_list.append(bfs_connected_list)
    return dfs_list, bfs_list


def main():

    # 将输入数据保存为列表
    n, m = map(int, input().split())
    edges = [tuple(map(int, input().split())) for _ in range(m)]

    # 建立图
    graph = defaultdict(list)
    for u, v in edges:
        graph[u].append(v)  # 由于是无向图，所以两端各需要保存一次
        graph[v].append(u)

    # 查找图中的连通集
    dfs_list, bfs_list = find_connected_sets(graph, n)

    # 结构化输出
    for data in dfs_list:
        print(f"{{ {' '.join(map(str, data))} }}")
    for data in bfs_list:
        print(f"{{ {' '.join(map(str, data))} }}")


if __name__ == "__main__":
    main()
```

【整体思路】

1. 读取输入数据，包括节点数量n、边数量m以及各边的两个端点
2. 构建无向图的邻接表表示
3. 应用DFS和BFS算法分别查找所有连通分量
4. 按格式要求输出结果

【关键点解析1】

```
# 通过排序实现按编号递增的顺序访问邻接点,由于栈后进先出，所以要降序
            stack.extend(sorted(graph[vertex], reverse=True))
```

- 题目中要求“按编号递增的顺序访问邻接点”，这意味着在放入堆栈和队列前先要对结点进行排序
- sorted(graph[vertex], reverse=True)
    - 使用python内置的sorted函数，先获取一个可迭代对象，然后将关键字设置为True——表明要降序排，否则默认为升序排列
- stack.extend方法可以将一组数据存入列表中，与append方法相近

【关键点解析2】

```
    visited_dfs = set()
    visited_bfs = set()
```

- 使用set存储已访问节点是有优势的，与list相比
    - 查找效率更高：在列表中检查元素是否存在（if vertex not in visited）的时间复杂度是 O(n)，需要遍历整个列表。而在集合中这个操作的时间复杂度只有 O(1)，因为集合基于哈希表实现
    - 避免性能瓶颈：在图遍历过程中，我们需要频繁检查节点是否已被访问。如果使用列表，随着已访问节点数量增加，每次检查的时间也会增加，导致整个算法的时间复杂度劣化到 $O(n^2)$。
- 为DFS和BFS分别维护独立的已访问集合，确保两种算法互不干扰

【关键点解析3】

```
from collections import defaultdict, deque

# 建立图
    graph = defaultdict(list)
    for u, v in edges:
        graph[u].append(v)  # 由于是无向图，所以两端各需要保存一次
        graph[v].append(u)
```

- 双端队列的用法之前已经讲过，见Python中的双端队列（deque）
- defaultdict也是python编程优雅性的体现
    - defaultdict 是 dict 的一个子类，它在创建时需要传入一个工厂函数（通常是一个可调用对象，像内置的数据类型函数，如 list、int、set 等）。当访问字典中不存在的键时，defaultdict 会自动使用这个工厂函数为该键创建一个默认值，而不是像普通字典那样抛出 KeyError 异常
    - 在 graph = defaultdict(list) 中，工厂函数是 list。这意味着当你访问 graph 中一个不存在的键时，defaultdict 会自动为这个键创建一个空列表作为默认值
    - 如果不用这种方式，常规的写法需要增加检查语句，如下所示

```
graph = {}
u, v = 0, 1
# 检查顶点 u 是否存在于字典中
if u not in graph:
    graph[u] = []
# 检查顶点 v 是否存在于字典中
if v not in graph:
    graph[v] = []
graph[u].append(v)
graph[v].append(u)
```