Assignment #B: 图论和树算

Updated 1709 GMT+8 Apr 28, 2024

2024 spring, Complied by ==同学的姓名、院系==

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 Ventura 13.4.1 (c)

Python编程环境：Spyder IDE 5.2.2, PyCharm 2023.1.4 (Professional Edition)

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

28170: 算鹰

dfs, http://cs101.openjudge.cn/practice/28170/

思路：

dfs寻找联通区域数量

代码

```
from itertools import product
WIDTH = 10
DIRECTIONS = [(1, 0), (-1, 0), (0, 1), (0, -1)]
board = [[1 if x == '.' else 0 for x in input()] for _ in range(WIDTH)]
traversed = [[0] * WIDTH for _ in range(WIDTH)]
```

```python
def is_out_of_bound(point):
    x, y = point
    return x < 0 or x >= WIDTH or y < 0 or y >= WIDTH


def dfs(point):
    x, y = point
    if traversed[x][y]:
        return
    traversed[x][y] = 1
    for dx, dy in DIRECTIONS:
        new_point = (x + dx, y + dy)
        if not is_out_of_bound(new_point) and board[x][y]:
            dfs(new_point)


res = 0
for pointin in product(range(WIDTH), range(WIDTH)):
    xin, yin = pointin
    if not traversed[xin][yin] and board[xin][yin]:
        res += 1
        dfs(pointin)
print(res)
```

代码运行截图 ==（至少包含有"Accepted"）==

状态: Accepted

源代码

```
# 28170:算鹰
from itertools import product
WIDTH = 10
DIRECTIONS = [(1, 0), (-1, 0), (0, 1), (0, -1)]
board = [[1 if x == '.' else 0 for x in input()] for _ in range(WIDTH)]
traversed = [[0] * WIDTH for _ in range(WIDTH)]


def is_out_of_bound(point):
    x, y = point
    return x < 0 or x >= WIDTH or y < 0 or y >= WIDTH


def dfs(point):
    x, y = point
    if traversed[x][y]:
        return
    traversed[x][y] = 1
    for dx, dy in DIRECTIONS:
        new_point = (x + dx, y + dy)
        if not is_out_of_bound(new_point) and board[x][y]:
            dfs(new_point)


res = 0
for pointin in product(range(WIDTH), range(WIDTH)):
    xin, yin = pointin
    if not traversed[xin][yin] and board[xin][yin]:
        res += 1
        dfs(pointin)
print(res)
```

02754: 八皇后

dfs, http://cs101.openjudge.cn/practice/02754/

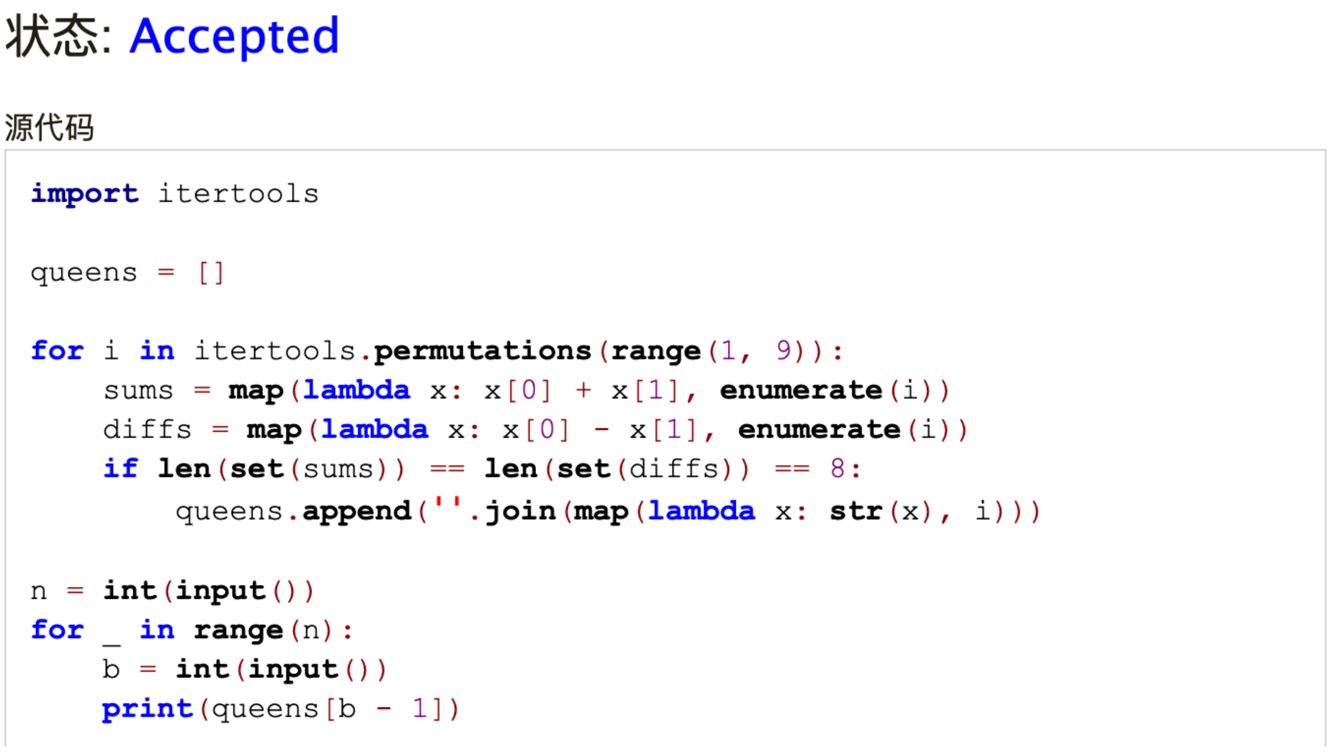
思路：

dfs

代码

```python
import itertools

queens = []

for i in itertools.permutations(range(1, 9)):
    sums = map(lambda x: x[0] + x[1], enumerate(i))
    diffs = map(lambda x: x[0] - x[1], enumerate(i))
    if len(set(sums)) == len(set(diffs)) == 8:
        queens.append(''.join(map(lambda x: str(x), i)))

n = int(input())
for _ in range(n):
    b = int(input())
    print(queens[b - 1])
```

代码运行截图 ==（至少包含有"Accepted"）==


状态: Accepted

源代码

```python
import itertools

queens = []

for i in itertools.permutations(range(1, 9)):
    sums = map(lambda x: x[0] + x[1], enumerate(i))
    diffs = map(lambda x: x[0] - x[1], enumerate(i))
    if len(set(sums)) == len(set(diffs)) == 8:
        queens.append(''.join(map(lambda x: str(x), i)))

n = int(input())
for _ in range(n):
    b = int(input())
    print(queens[b - 1])
```

03151: Pots

bfs, http://cs101.openjudge.cn/practice/03151/

思路：

bfs。检测到成功就跳出循环

代码

```python
# 03151:Pots
from collections import deque


def fill1(state):
    return (VOLUME[0], state[1])


def fill2(state):
    return (state[0], VOLUME[1])


def drop1(state):
    return (0, state[1])


def drop2(state):
    return (state[0], 0)


def pour12(state):
    if state[0] > VOLUME[1] - state[1]:
        return (state[0] + state[1] - VOLUME[1], VOLUME[1])
    return (0, state[0] + state[1])


def pour21(state):
    if state[1] > VOLUME[0] - state[0]:
        return (VOLUME[0], state[1] + state[0] - VOLUME[0])
    return (state[0] + state[1], 0)


def find_path(state, depth=0):
    if traversed[state] == ():
        print(depth)
        return
    find_path(traversed[state][0], depth + 1)
    print(traversed[state][1])


METHODS = {fill1: 'FILL(1)', fill2: 'FILL(2)', drop1: 'DROP(1)',
           drop2: 'DROP(2)', pour12: 'POUR(1,2)', pour21: 'POUR(2,1)'}

*VOLUME, TARGET = map(int, input().split())
initial_state = (0, 0)
traversed = {initial_state: ()}
queue = deque()
queue.append(initial_state)
res = None
while len(queue) > 0:
    this = queue.popleft()
    for method, op in METHODS.items():
        now = method(this)
        if now not in traversed:
            traversed[now] = (this, op)
            queue.append(now)
            if TARGET in now:
                res = now
                break
    if res is not None:
        break

if res is None:
    print('impossible')
else:
    find_path(res)
```

代码运行截图 ==（AC代码截图，至少包含有"Accepted"）==

状态: Accepted

基本

源代码

```python
# 03151:Pots
from collections import deque


def fill1(state):
    return (VOLUME[0], state[1])


def fill2(state):
    return (state[0], VOLUME[1])


def drop1(state):
    return (0, state[1])


def drop2(state):
    return (state[0], 0)


def pour12(state):
    if state[0] > VOLUME[1] - state[1]:
        return (state[0] + state[1] - VOLUME[1], VOLUME[1])
    return (0, state[0] + state[1])


def pour21(state):
    if state[1] > VOLUME[0] - state[0]:
        return (VOLUME[0], state[1] + state[0] - VOLUME[0])
    return (state[0] + state[1], 0)


def find_path(state, depth=0):
    if traversed[state] == ():
        print(depth)
        return
    find_path(traversed[state][0], depth + 1)
    print(traversed[state][1])


METHODS = {fill1: 'FILL(1)', fill2: 'FILL(2)', drop1: 'DROP(1)',
           drop2: 'DROP(2)', pour12: 'POUR(1,2)', pour21: 'POUR(2,1)'}

*VOLUME, TARGET = map(int, input().split())
```

提

05907: 二叉树的操作

http://cs101.openjudge.cn/practice/05907/

思路：

简单的树的操作。

代码

```python
# 05907:二叉树的操作
class Node:
    def __init__(self, key):
        self.key = key
        self.father = None
        self.position = None
        self.children = ()

    def traverse(self):
        if len(self.children) > 0 and self.children[0] is not None:
            return self.children[0].traverse()
        return self


class Tree:
    def __init__(self, number_of_nodes):
        self.nodes = {key: Node(key) for key in range(number_of_nodes)}

    def _node_construct(self, key):
        if key in self.nodes:
            return self.nodes[key]
        return None

    def add_child(self, fkey, lkey, rkey):
        father, lchild, rchild = map(self._node_construct, (fkey, lkey, rkey))
        if father is not None:
            father.children = (lchild, rchild)
        if lchild is not None:
            lchild.father = father
            lchild.position = 0
        if rchild is not None:
            rchild.father = father
            rchild.position = 1

    @staticmethod
    def change_binary_tuple(t, position, x):
        if position == 0:
            return (x, t[1])
        if position == 1:
            return (t[0], x)
        return None
    def swap(self, key1, key2):
        node1, node2 = self._node_construct(key1), self._node_construct(key2)
        node1.father.children = Tree.change_binary_tuple(
            node1.father.children, node1.position, node2)
        node2.father.children = Tree.change_binary_tuple(
            node2.father.children, node2.position, node1)
        node1.father, node2.father = node2.father, node1.father
        node1.position, node2.position = node2.position, node1.position

    def traverse(self, key):
        return self._node_construct(key).traverse().key


t = int(input())
for _ in range(t):
    n, m = map(int, input().split())
    tree = Tree(n)
    for _ in range(n):
        x, y, z = map(int, input().split())
        tree.add_child(x, y, z)
    for _ in range(m):
        inp_type, *inp = map(int, input().split())
        if inp_type == 1:
            tree.swap(*inp)
        if inp_type == 2:
            print(tree.traverse(inp[0]))
```

代码运行截图 ==（AC代码截图，至少包含有"Accepted"）==

状态: Accepted

源代码

```
# 05907:二叉树的操作
class Node:
    def __init__(self, key):
        self.key = key
        self.father = None
        self.position = None
        self.children = ()

    def traverse(self):
        if len(self.children) > 0 and self.children[0] is not None:
            return self.children[0].traverse()
        return self


class Tree:
    def __init__(self, number_of_nodes):
        self.nodes = {key: Node(key) for key in range(number_of_nodes)}

    def _node_construct(self, key):
        if key in self.nodes:
            return self.nodes[key]
        return None

    def add_child(self, fkey, lkey, rkey):
        father, lchild, rchild = map(self._node_construct, (fkey, lkey,
        if father is not None:
            father.children = (lchild, rchild)
        if lchild is not None:
            lchild.father = father
            lchild.position = 0
        if rchild is not None:
            rchild.father = father
            rchild.position = 1

    @staticmethod
    def change_binary_tuple(t, position, x):
        if position == 0:
            return (x, t[1])
        if position == 1:
            return (t[0], x)
        return None

    def swap(self, key1, key2):
        node1, node2 = self._node_construct(key1), self._node_construct
```

18250: 冰阔落 I

Disjoint set, http://cs101.openjudge.cn/practice/18250/

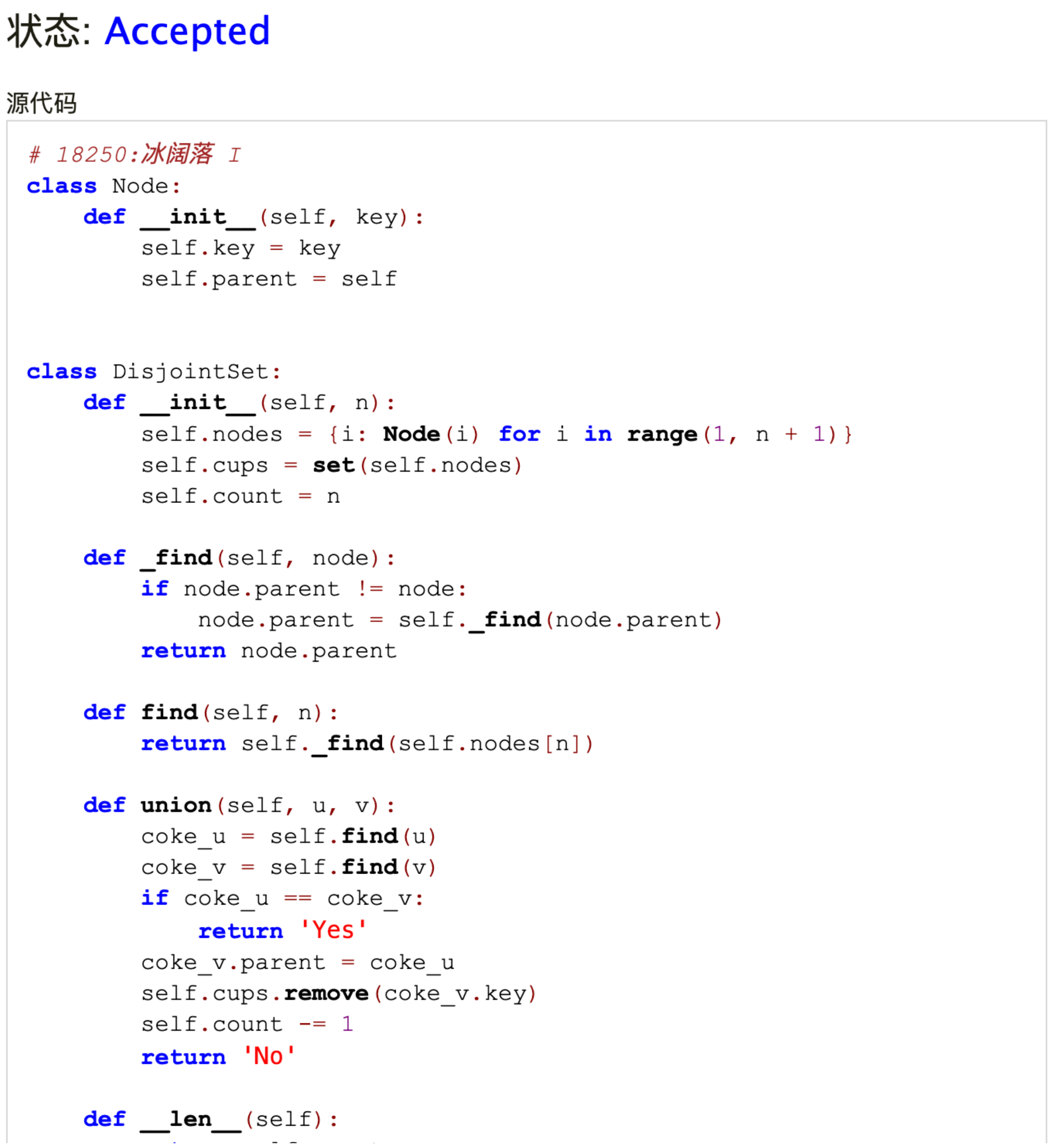
思路：

并查集的使用，注意读题。一开始读错了题目导致WA。

代码

```python
class Node:
    def __init__(self, key):
        self.key = key
        self.parent = self


class DisjointSet:
    def __init__(self, n):
        self.nodes = {i: Node(i) for i in range(1, n + 1)}
        self.cups = set(self.nodes)
        self.count = n

    def _find(self, node):
        if node.parent != node:
            node.parent = self._find(node.parent)
        return node.parent

    def find(self, n):
        return self._find(self.nodes[n])

    def union(self, u, v):
        coke_u = self.find(u)
        coke_v = self.find(v)
        if coke_u == coke_v:
            return 'Yes'
        coke_v.parent = coke_u
        self.cups.remove(coke_v.key)
        self.count -= 1
        return 'No'

    def __len__(self):
        return self.count

    def check_empty(self):
        return ' '.join(map(str, sorted(self.cups)))


while 1:
    try:
        nn, m = map(int, input().split())
        cups = DisjointSet(nn)
        for _ in range(m):
            x, y = map(int, input().split())
            print(cups.union(x, y))
        print(len(cups))
        print(cups.check_empty())
    except EOFError:
        break
```

代码运行截图 ==（AC代码截图，至少包含有"Accepted"）==



状态: Accepted

源代码

```python
# 18250:冰阔落 I
class Node:
    def __init__(self, key):
        self.key = key
        self.parent = self


class DisjointSet:
    def __init__(self, n):
        self.nodes = {i: Node(i) for i in range(1, n + 1)}
        self.cups = set(self.nodes)
        self.count = n

    def _find(self, node):
        if node.parent != node:
            node.parent = self._find(node.parent)
        return node.parent

    def find(self, n):
        return self._find(self.nodes[n])

    def union(self, u, v):
        coke_u = self.find(u)
        coke_v = self.find(v)
        if coke_u == coke_v:
            return 'Yes'
        coke_v.parent = coke_u
        self.cups.remove(coke_v.key)
        self.count -= 1
        return 'No'

    def __len__(self):
            return self.count

        def check_empty(self):
            return ' '.join(map(str, sorted(self.cups)))


    while 1:
        try:
            nn, m = map(int, input().split())
            cups = DisjointSet(nn)
            for _ in range(m):
                x, y = map(int, input().split())
                print(cups.union(x, y))
            print(len(cups))
            print(cups.check_empty())
        except EOFError:
            break
```

05443: 兔子与樱花

http://cs101.openjudge.cn/practice/05443/

思路：

Dijkstra算法，同时追加了输出最短路径的实现。

代码

```
import heapq

INFINITY = float('inf')


class Graph:
    def __init__(self, vertices):
        self.vertices = vertices
        self.connected_vertex = {vertex: [] for vertex in self.vertices}
        self.num_vertices = len(self.vertices)

    def connect(self, point1, point2, weight):
        self.connected_vertex[point1].append((point2, weight))
        self.connected_vertex[point2].append((point1, weight))

    def dijkstra(self, start, end):
        if start == end:
            return start
        visited = set()
        heap = [(0, start)]
        distance = {vertex: INFINITY for vertex in self.vertices}
        distance[start] = 0
        paths = {}
        while len(visited) < self.num_vertices and len(heap) > 0:
            now = heapq.heappop(heap)[1]
            if now in visited:
                continue
            visited.add(now)
            for point, edge_weight in self.connected_vertex[now]:
                if point not in visited and distance[now] + edge_weight <=
distance[point]:
                    distance[point] = distance[now] + edge_weight
                    heapq.heappush(heap, (distance[point], point))
                    paths[point] = (now, f'({edge_weight})')

        def find(end):
            if end == start:
                return []
            return [*find(paths[end][0]), *paths[end]]
        return '->'.join(find(end) + [end])


p = int(input())
vertices = [input() for _ in range(p)]
graph = Graph(vertices)
q = int(input())
for _ in range(q):
    point1, point2, weight = input().split()
    graph.connect(point1, point2, int(weight))
r = int(input())
for _ in range(r):
    print(graph.dijkstra(*input().split()))
```

代码运行截图 ==（AC代码截图，至少包含有"Accepted"）==

状态: Accepted

源代码

```python
# 05443:兔子与樱花
import heapq

INFINITY = float('inf')


class Graph:
    def __init__(self, vertices):
        self.vertices = vertices
        self.connected_vertex = {vertex: [] for vertex in self.vertices
        self.num_vertices = len(self.vertices)

    def connect(self, point1, point2, weight):
        self.connected_vertex[point1].append((point2, weight))
        self.connected_vertex[point2].append((point1, weight))

    def dijkstra(self, start, end):
        if start == end:
            return start
        visited = set()
        heap = [(0, start)]
        distance = {vertex: INFINITY for vertex in self.vertices}
        distance[start] = 0
        paths = {}
        while len(visited) < self.num_vertices and len(heap) > 0:
            now = heapq.heappop(heap)[1]
            if now in visited:
                continue
            visited.add(now)
            for point, edge_weight in self.connected_vertex[now]:
                if point not in visited and distance[now] + edge_weight
                    distance[point] = distance[now] + edge_weight
                    heapq.heappush(heap, (distance[point], point))
                    paths[point] = (now, f'({edge_weight})')

        def find(end):
            if end == start:
                return []
            return [*find(paths[end][0]), *paths[end]]
        return '->'.join(find(end) + [end])


p = int(input())
```

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。==

复习了并查集，dijkstra算法。