Assignment #B: 图论和树算

Updated 1709 GMT+8 Apr 28, 2024

2024 spring, Complied by 胡登科

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：熟练掌握了dfs算法，就是一直走然后回溯，这个题目对于ct的改编要注意不要走错点了。、

```
field = [list(input()) for _ in range(n)]
通过这句话将输入数据转化为二维数组。
```

代码

```
#
# 1.dfs
import sys

sys.setrecursionlimit(20000)


def dfs(x, y):
    # 标记，避免再次访问
```

```
    field[x][y] = '-'
    for k in range(4):
        nx, ny = x + dx[k], y + dy[k]
        # 范围内且未访问的lake
        if 0 <= nx < n and 0 <= ny < m \
                and field[nx][ny] == '.':
            # 继续搜索
            dfs(nx, ny)


(n, m) = (10, 10)
field = [list(input()) for _ in range(n)]
cnt = 0
dx = [0, -1, 1, 0]
dy = [-1, 0, 0, 1]
for i in range(n):
    for j in range(m):
        if field[i][j] == '.':
            dfs(i, j)
            cnt += 1
print(cnt)
```

代码运行截图 （至少包含有"Accepted"）

状态: Accepted

源代码

```
# 1.dfs
import sys

sys.setrecursionlimit(20000)


def dfs(x, y):
    # 标记，避免再次访问
    field[x][y] = '-'
    for k in range(4):
        nx, ny = x + dx[k], y + dy[k]
        # 范围内且未访问的lake
        if 0 <= nx < n and 0 <= ny < m \
                and field[nx][ny] == '.':
            # 继续搜索
            dfs(nx, ny)


(n, m) = (10, 10)
field = [list(input()) for _ in range(n)]
cnt = 0
dx = [0, -1, 1, 0]
dy = [-1, 0, 0, 1]
for i in range(n):
    for j in range(m):
        if field[i][j] == '.':
            dfs(i, j)
            cnt += 1
print(cnt)
```


02754: 八皇后

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

思路:

仔细思考了八皇后问题，看到一个简单易懂的代码。通过第一个函数进行挖掘，将所有挖掘到的可行结果放在solution中。这个函数是自然回溯的过程，直接通过不断地往下搜索row的可能性。f(n-1)i的情况下搜索fn1-8，得到fnj，递归栈存储fnj，然后将fnj进一步搜索f(n+1)k，到达f8即可。然后一次次的打出栈就是回溯的过程。‘

之前觉得这种题目挺难的，但是明白dfs的逻辑后觉得思考过程更加清晰了。

代码

```
# 八皇后
def solve_n_queens(n):
    stack = []
    solutions = []
```

```
    stack.append((0, [-1] * n))

    while stack:
        row, queens = stack.pop()
        if row == n:
            solutions.append(queens.copy())
        else:
            for col in range(n):
                if is_valid(row, col, queens):
                    new_queens = queens.copy()
                    new_queens[row] = col
                    stack.append((row + 1, new_queens))

    return solutions


def is_valid(row, col, queens):
    for r in range(row):
        if queens[r] == col or abs(row - r) == abs(col - queens[r]):
            return False
    return True


def get_queen_string(b):
    solutions = solve_n_queens(8)
    if b > len(solutions):
        return None
    b = len(solutions) + 1 - b
    queen_string = ''.join(str(col + 1) for col in solutions[b - 1])
    return queen_string


test_cases = int(input())
for _ in range(test_cases):
    b = int(input())
    queen_string = get_queen_string(b)
    print(queen_string)
```

代码运行截图 （至少包含有"Accepted"）

状态: Accepted

源代码

```
# 八皇后
def solve_n_queens(n):
    stack = []
    solutions = []

    stack.append((0, [-1] * n))

    while stack:
        row, queens = stack.pop()
        if row == n:
            solutions.append(queens.copy())
        else:
            for col in range(n):
                if is_valid(row, col, queens):
                    new_queens = queens.copy()
                    new_queens[row] = col
                    stack.append((row + 1, new_queens))

    return solutions


def is_valid(row, col, queens):
    for r in range(row):
        if queens[r] == col or abs(row - r) == abs(col - queens[r]):
            return False
    return True


def get_queen_string(b):
```

03151: Pots

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

思路：# pots 最短路径问题 通常使用bfs bfs就是自己去假设下一次会发生的所有可能性 全部储存在visited中

感觉bfs比dfs在代码上困难。

代码

```
#
# pots 最短路径问题 通常使用bfs bfs就是自己去假设下一次会发生的所有可能性 全部储存在visited中
def bfs(A, B, C):
    start = (0, 0)
    visited = set()
    visited.add(start)
    queue = [(start, [])]

    while queue:
        (a, b), actions = queue.pop(0)
```

```
        if a == C or b == C:
            return actions

        next_states = [(A, b), (a, B), (0, b), (a, 0), (min(a + b, A), max(0, a +
b - A)), (max(0, a + b - B), min(a + b, B))]

        for i in next_states:
            if i not in visited:
                visited.add(i)
                new_actions = actions + [get_action(a, b, i)]
                queue.append((i, new_actions))

    return ["impossible"]


def get_action(a, b, next_state):
    if next_state == (A, b):
        return "FILL(1)"
    elif next_state == (a, B):
        return "FILL(2)"
    elif next_state == (0, b):
        return "DROP(1)"
    elif next_state == (a, 0):
        return "DROP(2)"
    elif next_state == (min(a + b, A), max(0, a + b - A)):
        return "POUR(2,1)"
    else:
        return "POUR(1,2)"


A, B, C = map(int, input().split())
solution = bfs(A, B, C)

if solution == ["impossible"]:
    print(solution[0])
else:
    print(len(solution))
    for i in solution:
        print(i)
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）

状态: Accepted

基本信息

#: 44886458
题目: 03151
提交人: 2200012286 胡登科
内存: 3704kB
时间: 21ms
语言: Python3
提交时间: 2024-05-07 11:07:10

源代码

```
# pots 最短路径问题 通常使用bfs bfs就是自己去假设下一次会发生的所有可能性 全部储存
def bfs(A, B, C):
    start = (0, 0)
    visited = set()
    visited.add(start)
    queue = [(start, [])]

    while queue:
        (a, b), actions = queue.pop(0)

        if a == C or b == C:
            return actions

        next_states = [(A, b), (a, B), (0, b), (a, 0), (min(a + b, A), r

        for i in next_states:
            if i not in visited:
                visited.add(i)
                new_actions = actions + [get_action(a, b, i)]
                queue.append((i, new_actions))

    return ["impossible"]


def get_action(a, b, next_state):
    if next_state == (A, b):
        return "FILL(1)"
    elif next_state == (a, B):
        return "FILL(2)"
    elif next_state == (0, b):
```

05907: 二叉树的操作

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

思路：建树，每一个节点有四个属性。val，前驱指针，左子结点指针，右子节点指针。所有节点保存在一个lst中。交换：通过val查询前驱指针下的node对应的val对应的子节点指针。获取两个要交换的子节点和其前驱，交换两次，就能完成交换。前驱查询：化简版的交换。

代码

```
# # 二叉树的操作 常规树的问题 格式很重要
class TreeNode:
    def __init__(self, val=0):
        self.val = val
        self.left = None
        self.right = None
        self.parent = None


def build_tree(nodes_info):
    nodes = [TreeNode(i) for i in range(n)]
    for val, left, right in nodes_info:
        if left != -1:
            nodes[val].left = nodes[left]
        if right != -1:
            nodes[val].right = nodes[right]
    return nodes
```

```
def swap_nodes(nodes, x, y):
    for node in nodes:
        if node.left and node.left.val in [x, y]:
            node.left = nodes[y] if node.left.val == x else nodes[x]
        if node.right and node.right.val in [x, y]:
            node.right = nodes[y] if node.right.val == x else nodes[x]


def find_leftmost(node):
    while node and node.left:
        node = node.left
    return node.val if node else -1


for _ in range(int(input())):
    n, m = map(int, input().split())
    nodes_info = [tuple(map(int, input().split())) for _ in range(n)]
    ops = [tuple(map(int, input().split())) for _ in range(m)]
    nodes = build_tree(nodes_info)
    for op in ops:
        if op[0] == 1:
            swap_nodes(nodes, op[1], op[2])
        elif op[0] == 2:
            print(find_leftmost(nodes[op[1]]))
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）

#44886671提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
# 二叉树的操作 常规树的问题 格式很重要
class TreeNode:
    def __init__(self, val=0):
        self.val = val
        self.left = None
        self.right = None


def build_tree(nodes_info):
    nodes = [TreeNode(i) for i in range(n)]
    for val, left, right in nodes_info:
        if left != -1:
            nodes[val].left = nodes[left]
        if right != -1:
            nodes[val].right = nodes[right]
    return nodes


def swap_nodes(nodes, x, y):
    for node in nodes:
        if node.left and node.left.val in [x, y]:
            node.left = nodes[y] if node.left.val == x else nodes[x]
        if node.right and node.right.val in [x, y]:
            node.right = nodes[y] if node.right.val == x else nodes[x]
```

基本信息

#: 44886671
题目: 05907
提交人: 2200012286 胡登科
内存: 3816kB
时间: 155ms
语言: Python3
提交时间: 2024-05-07 11:31:40

18250: 冰阔落 I

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

思路：用一个lst存储我们的杯子，用4个值定义杯子name=i，val=1，pointer1，pointer2。初始化均为None。倒可乐后面数的pointer1指向前一个name，前一个name的pointer2指向后一个name，后一个的val清0，前一个+1。两个name查询第一个name的pointer1对应的name的pointer1直到pointer1 = None。查询第一个name的pointer2直到pointer2 = None ，当找到后面name为止，返回Yes。找到None 就返回No。

这个思路可以不用并查集。

代码

```
# 这个代码只是一个思路 没有debug  后面有用并查集做的方法
class glass:
    def __init__(self, name):
        self.name = name
        self.val = 1
        self.pointer1 = None
        self.pointer2 = None


def pour(name1, name2):
    glasses[name2].pointer1 = glasses[name1]
    glasses[name1].pointer2 = glasses[name2]
    glasses[name1].val += glasses[name2].val
    glasses[name2].val = 0


def check(name1, name2):
    find = find_None(name1, name2)
    if find == 'Not find':
        print("No")
        pour(name1, name2)
    else:
        print("Yes")


def find_None(name, target):
    if glasses[name].pointer1.pointer1 == None:
        return "Not find"
    find_None(glasses[name].pointer1, target)

    if glasses[name].pointer2.name == None:
        return "Not find"
    find_None(glasses[name].pointer2, target)

    return "find"


glasses = []
count = 0
lst = []
```

```
n, m = map(int, input().split())
for _ in range(n):
    glasses.append(glass(_))
for _ in range(m):
    x, y = map(int, input().split())
    check(x, y)
for item in glasses:
    if item.val != 0:
        count += 1
        lst.append(item.name)
print(count)
for _ in range(count):
    print(lst[_])
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）

#44886909提交状态

状态: Accepted

源代码

```
def find(x):
    if parent[x] != x:
        parent[x] = find(parent[x])
    return parent[x]

def union(x, y):
    root_x = find(x)
    root_y = find(y)
    if root_x != root_y:
        parent[root_y] = root_x

while True:
    try:
        n, m = map(int, input().split())
        parent = list(range(n + 1))

        for _ in range(m):
            a, b = map(int, input().split())
            if find(a) == find(b):
                print('Yes')
            else:
```

05443: 兔子与樱花

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

思路：最短路径dijkstra。对于这个算法理解还不到位，看也没看明白。

代码

```
#
# 兔子题 最短路径 dijkstra
import heapq

def dijkstra(adjacency, start):
    distances = {vertex: float('infinity') for vertex in adjacency}
    previous = {vertex: None for vertex in adjacency}
    distances[start] = 0
    pq = [(0, start)]

    while pq:
        current_distance, current_vertex = heapq.heappop(pq)
        if current_distance > distances[current_vertex]:
            continue

        for neighbor, weight in adjacency[current_vertex].items():
            distance = current_distance + weight
            if distance < distances[neighbor]:
                distances[neighbor] = distance
                previous[neighbor] = current_vertex
                heapq.heappush(pq, (distance, neighbor))

    return distances, previous

def shortest_path_to(adjacency, start, end):
    distances, previous = dijkstra(adjacency, start)
    path = []
    current = end
    while previous[current] is not None:
        path.insert(0, current)
        current = previous[current]
    path.insert(0, start)
    return path, distances[end]

# Read the input data
P = int(input())
places = {input().strip() for _ in range(P)}

Q = int(input())
graph = {place: {} for place in places}
for _ in range(Q):
    src, dest, dist = input().split()
    dist = int(dist)
    graph[src][dest] = dist
    graph[dest][src] = dist  # Assuming the graph is bidirectional

R = int(input())
requests = [input().split() for _ in range(R)]

# Process each request
for start, end in requests:
    if start == end:
        print(start)
```

```
        continue

    path, total_dist = shortest_path_to(graph, start, end)
    output = ""
    for i in range(len(path) - 1):
        output += f"{path[i]}->({graph[path[i]][path[i+1]]})->"
    output += f"{end}"
    print(output)
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）

#44886917提交状态

状态: Accepted

源代码

```
# 兔子题 最短路径 dijkstra
import heapq

def dijkstra(adjacency, start):
    distances = {vertex: float('infinity') for vertex in adjacency}
    previous = {vertex: None for vertex in adjacency}
    distances[start] = 0
    pq = [(0, start)]

    while pq:
        current_distance, current_vertex = heapq.heappop(pq)
        if current_distance > distances[current_vertex]:
            continue

        for neighbor, weight in adjacency[current_vertex].items():
            distance = current_distance + weight
            if distance < distances[neighbor]:
                distances[neighbor] = distance
                previous[neighbor] = current_vertex
                heapq.heappush(pq, (distance, neighbor))

    return distances, previous

def shortest_path_to(adjacency, start, end):
    distances, previous = dijkstra(adjacency, start)
    path = []
    current = end
    while previous[current] is not None:
        path.insert(0, current)
        current = previous[current]
    path.insert(0, start)
    return path, distances[end]
```

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。

5.1节期间把其他的事情做的差不多了，是时候履行做数算的承诺了。对于题目思考变多了，开始看教材复习基础知识。加油。