Assignment #A: 图论：算法，树算及栈

Updated 2018 GMT+8 Apr 21, 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

20743: 整人的提词本

http://cs101.openjudge.cn/practice/20743/

思路：

简单的栈实现。

代码

```
stack = []
inp = input()
for char in inp:
    if char == ')':
        aux = []
        while stack[-1] != '(':
            aux.append(stack.pop())
        stack.pop()
        stack += aux
    else:
        stack.append(char)
print(''.join(stack))
```

代码运行截图 ==（至少包含有"Accepted"）==

状态: Accepted

源代码

```
#20743:整人的提词本
stack = []
inp = input()
for char in inp:
    if char == ')':
        aux = []
        while stack[-1] != '(':
            aux.append(stack.pop())
        stack.pop()
        stack += aux
    else:
        stack.append(char)
print(''.join(stack))
```

02255: 重建二叉树

http://cs101.openjudge.cn/practice/02255/

思路：

在之前的作业中出现过类似的题目。

代码

```
def parse(forward, backward):
    if len(forward) == 0:
        return ''
    if len(forward) == 1:
        return forward[0]
    root = forward[0]
    index = backward.find(root)
    return parse(forward[1:index + 1], backward[:index]) + parse(forward[index + 1:],
backward[index + 1:]) + root
while 1:
    try:
        forward, backward = input().split()
    except EOFError:
        break
    print(parse(forward, backward))
```

代码运行截图 ==（至少包含有"Accepted"）==

#44830006提交状态

状态: Accepted

源代码

```
#02255:重建二叉树
def parse(forward, backward):
    if len(forward) == 0:
        return ''
    if len(forward) == 1:
        return forward[0]
    root = forward[0]
    index = backward.find(root)
    return parse(forward[1:index + 1], backward[:index]) + parse(forwar
while 1:
    try:
        forward, backward = input().split()
    except EOFError:
        break
    print(parse(forward, backward))
```

01426: Find The Multiple

http://cs101.openjudge.cn/practice/01426/

要求用bfs实现

思路：

bfs。

代码

```
from queue import deque
while 1:
    num = int(input())
    if num == 0:
        break
    queue = deque([1])
    while len(queue) > 0:
        i = queue.popleft()
        if i % num == 0:
            print(i)
            break
        queue.append(10 * i)
        queue.append(10 * i + 1)
```

代码运行截图 ==（AC代码截图，至少包含有"Accepted"）==

状态: Accepted

源代码

```
#01426:Find The Multiple
from queue import deque
while 1:
    num = int(input())
    if num == 0:
        break
    queue = deque([1])
    while len(queue) > 0:
        i = queue.popleft()
        if i % num == 0:
            print(i)
            break
        queue.append(10 * i)
        queue.append(10 * i + 1)
```

04115: 鸣人和佐助

bfs, http://cs101.openjudge.cn/practice/04115/

思路：

记录在经过每个点时剩余的查克拉量，我的代码可以正常通过题目中的样例和群内构造的样例，但是WA，苦于没有测试用例，debug较难，最终借鉴了群里大佬的代码。

代码

```
import sys
sys.setrecursionlimit(100000)

m, n, t = map(int, input().split())
l = [[-1] * n for _ in range(m)]
move_offset = [(0, -1), (0, 1), (-1, 0), (1, 0)]
pos1 = ()
pos2 = ()
ll = []
for y in range(m):
    temp = input()
    if '@' in temp:
        x = temp.index('@')
        pos1 = (x, y)
        l[y][x] = t
    if '+' in temp:
        x = temp.index('+')
        pos2 = (x, y)
    ll.append(temp)

def bfs(lx:list, steps:int = 0):
    global pos2, move_offset, ll, l, n, m
    if lx == []:
        return -1
    l2 = []
    for info in lx:
        pos, t = info
        if pos == pos2:
            return steps
        else:
            x, y = pos
            for offset in move_offset:
                i, j = offset
                xi, yj, tt = x + i, y + j, t
                if xi >= 0 and yj >= 0 and xi < n and yj < m:
                    if ll[yj][xi] == '#':
                        tt -= 1
                    if tt >= 0 and l[yj][xi] < tt:
```

```
                        l[yj][xi] = tt
                        l2.append(((xi, yj), tt))
    return bfs(l2, steps + 1)

print(bfs([(pos1, t)]))
```

代码运行截图 ==（AC代码截图，至少包含有"Accepted"）==

状态: Accepted

源代码

```
import sys
sys.setrecursionlimit(100000)

m, n, t = map(int, input().split())
l = [[-1] * n for _ in range(m)]
move_offset = [(0, -1), (0, 1), (-1, 0), (1, 0)]
pos1 = ()
pos2 = ()
ll = []
for y in range(m):
    temp = input()
    if '@' in temp:
        x = temp.index('@')
        pos1 = (x, y)
        l[y][x] = t
    if '+' in temp:
        x = temp.index('+')
        pos2 = (x, y)
    ll.append(temp)

def bfs(lx:list, steps:int = 0):
    global pos2, move_offset, ll, l, n, m
    if lx == []:
        return -1
    l2 = []
    for info in lx:
        pos, t = info
        if pos == pos2:
            return steps
        else:
            x, y = pos
            for offset in move_offset:
                i, j = offset
                xi, yj, tt = x + i, y + j, t
                if xi >= 0 and yj >= 0 and xi < n and yj < m:
                    if ll[yj][xi] == '#':
                        tt -= 1
                    if tt >= 0 and l[yj][xi] < tt:
                        l[yj][xi] = tt
                        l2.append(((xi, yj), tt))
    return bfs(l2, steps + 1)

print(bfs([(pos1, t)]))
```

我的WA代码

```
# 04115:鸣人和佐助
from collections import deque

m, n, t = map(int, input().split())
traversed = set()
bfs = deque()
directions = [(0, 1), (0, -1), (1, 0), (-1, 0)]
grid = [[-1 for _ in range(n)] for _ in range(m)]
value = [[-1 for _ in range(n)] for _ in range(m)]
time = [[-1 for _ in range(n)] for _ in range(m)]

for i in range(m):
    for j, char in enumerate(input()):
        if char in ('*', '+', '@'):
            grid[i][j] = 0
        if char == '+':
            start = (i, j)
            value[i][j] = t
            time[i][j] = 0
        if char == '@':
            end = (i, j)

traversed.add(start)
bfs.append(start)

while len(bfs) > 0:
    point = bfs.popleft()
    for dx, dy in directions:
        if point[0] + dx >= m or point[0] + dx < 0 or point[1] + dy >= n or point[1] + dy
< 0:
            continue

        next_move = (point[0] + dx, point[1] + dy)
        new_value = (value[point[0]][point[1]]
                     + grid[next_move[0]][next_move[1]])

        if new_value <= value[next_move[0]][next_move[1]] and next_move in traversed:
            continue

        if new_value > value[next_move[0]][next_move[1]]:
            bfs.append(next_move)
            value[next_move[0]][next_move[1]] = new_value
            if time[next_move[0]][next_move[1]] == -1:
                time[next_move[0]][next_move[1]] = time[point[0]][point[1]] + 1
            else:
                time[next_move[0]][next_move[1]] = min(
                    time[next_move[0]][next_move[1]], time[point[0]][point[1]] + 1)

        traversed.add(next_move)
```

```
print(time[end[0]][end[1]])
```

20106: 走山路

Dijkstra, http://cs101.openjudge.cn/practice/20106/

思路：

练习了Dijkstra算法，不使用heap会超时，善用heapq。

代码

```
# 20106:走山路
from itertools import product
import heapq

M, N, P = map(int, input().split())
INFINITY = float('inf')
MAP = [[int(x) if x != '#' else INFINITY for x in input().split()]
       for _ in range(M)]


def height(point):
    x, y = point
    return MAP[x][y]


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

    DIRECTIONS = [(0, -1), (-1, 0)]
    x, y = point
    res = []
    for dx, dy in DIRECTIONS:
        new_point = (x + dx, y + dy)
        if not is_out_of_bound(new_point):
            res.append(new_point)
    return res


def weight(point1, point2):
    return abs(height(point1) - height(point2))
```

```
class Graph:
    def __init__(self):
        self.connected_vertex = {(i, j): []
                                 for i, j in product(range(M), range(N))}
        for point in product(range(M), range(N)):
            for adj in adjacent_points(point):
                self.connect(point, adj)

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

    def dijkstra(self, start, end):
        visited = set()
        heap = [(0, start)]
        distance = {(i, j): INFINITY
                    for i, j in product(range(M), range(N))}
        distance[start] = 0
        while len(visited) < M*N 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))
        return distance[end] if distance[end] != INFINITY else 'NO'


graph = Graph()
for _ in range(P):
    x1, y1, x2, y2 = map(int, input().split())
    print(graph.dijkstra((x1, y1), (x2, y2)))
```

代码运行截图 ==（AC代码截图，至少包含有"Accepted"）==

状态: Accepted

源代码

```
# 20106:走山路
from itertools import product
import heapq

M, N, P = map(int, input().split())
INFINITY = float('inf')
MAP = [[int(x) if x != '#' else INFINITY for x in input().split()]
       for _ in range(M)]


def height(point):
    x, y = point
    return MAP[x][y]


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

    DIRECTIONS = [(0, -1), (-1, 0)]
    x, y = point
    res = []
    for dx, dy in DIRECTIONS:
        new_point = (x + dx, y + dy)
        if not is_out_of_bound(new_point):
            res.append(new_point)
    return res


def weight(point1, point2):
    return abs(height(point1) - height(point2))


class Graph:
    def __init__(self):
        self.connected_vertex = {(i, j): []
                                 for i, j in product(range(M), range(N))}
        for point in product(range(M), range(N)):
            for adj in adjacent_points(point):
                self.connect(point, adj)
```

05442: 兔子与星空

Prim, http://cs101.openjudge.cn/practice/05442/

思路：

学习了最小生成树的prim算法，并简单的实现了。

代码

```
# 05442:兔子与星空
import heapq
N = int(input())
vertices = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"[:N]
edges = {x: [] for x in vertices}

def connect(vertex1, vertex2, w):
    edges[vertex1].append((w, vertex2))
    edges[vertex2].append((w, vertex1))

for _ in range(N - 1):
    now, count, *others = map(lambda x: int(x) if x not in vertices else x,
input().split())
    for _ in range(count):
        other, weight = others.pop(0), others.pop(0)
        connect(now, other, weight)

res = 0
traversed = ['A']
vertices = set(vertices)
heap = []
for x in edges['A']:
    heapq.heappush(heap, x)
while len(traversed) < N and len(heap) > 0:
    minimum_edge = heapq.heappop(heap)
    if minimum_edge[1] not in traversed:
        res += minimum_edge[0]
        for x in edges[minimum_edge[1]]:
            heapq.heappush(heap, x)
        traversed.append(minimum_edge[1])
print(res)
```

代码运行截图 ==（AC代码截图，至少包含有"Accepted"）==

#44838309提交状态

状态: Accepted

源代码

```
# 05442:兔子与星空
import heapq
N = int(input())
vertices = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"[:N]
edges = {x: [] for x in vertices}

def connect(vertex1, vertex2, w):
    edges[vertex1].append((w, vertex2))
    edges[vertex2].append((w, vertex1))

for _ in range(N - 1):
    now, count, *others = map(lambda x: int(x) if x not in vertices else
    for _ in range(count):
        other, weight = others.pop(0), others.pop(0)
        connect(now, other, weight)

res = 0
traversed = ['A']
vertices = set(vertices)
heap = []
for x in edges['A']:
    heapq.heappush(heap, x)
while len(traversed) < N and len(heap) > 0:
    minimum_edge = heapq.heappop(heap)
    if minimum_edge[1] not in traversed:
        res += minimum_edge[0]
        for x in edges[minimum_edge[1]]:
            heapq.heappush(heap, x)
        traversed.append(minimum_edge[1])
print(res)
```

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。==

这周的作业中遇到了许多剪枝问题，同时学到了Dijkstra算法和Prim算法。