Assignment #A: 图论：算法，树算及栈

Updated 2018 GMT+8 Apr 21, 2024

2024 spring, Complied by 城环 吴至超

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：windows11

Python编程环境：pycharm2023.2.3

1. 题目

20743: 整人的提词本

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

思路：递归，从左到右，遇到“（”就递归，否则正常往当前的栈里加字母，遇到“）”就返回颠倒一次的结果，最后发现刚好函数输出结果与答案相反（因为最外一层还要倒一次），人为在输出那里再倒一次就好。

代码

```
# #20743:整人的提词本
s=list(input())

def reverse(a):
    stack=[]
    while s:
        tempo=s.pop(0)
        if tempo.isalpha():
            stack.append(tempo)
        elif tempo=="(":
            stack.extend(reverse(tempo))
        else:
            break
    return stack[::-1]
```

```
print("".join(reverse(None)[::-1]))
```

代码运行截图 ==（至少包含有"Accepted"）==

02255: 重建二叉树

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

思路：复习题，记得建树的两个参数都列表化，前序不断弹出作**根节点**（不断改变），不断传递，根据当前**根节点**在**中序遍历**中的位置确定好左右子树的中序遍历结果（原中序不变），递归即可。

代码

```
# #02255:重建二叉树
class tree:
    def __init__(self,name):
        self.name=name
        self.left=None
        self.right=None
def transfer(prelist,midlist):
    if not midlist:
        return None
    i=midlist.index(prelist[0])
    root=tree(prelist.pop(0))
    root.left=transfer(prelist,midlist[:i])
    root.right=transfer(prelist,midlist[i+1::])
    return root
def postorder(root):
    stack=[]
    if root.left:
        stack.extend(postorder(root.left))
    if root.right:
        stack.extend(postorder(root.right))
```

```
    stack.append(root)
    return stack
while True:
    try:
        preorder,midorder=map(str,input().split())
        prelist=list(preorder)
        midlist=list(midorder)
        postpri=postorder(transfer(prelist,midlist))
        for i in postpri:
            print(i.name,end="")
        print()
    except EOFError:
        break
```

代码运行截图 ==（至少包含有"Accepted"）==

01426: Find The Multiple

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

要求用bfs实现

思路：有一种办法是对*10 和 *10+1，借住余数代替表示int进行剪枝；

另一种办法是不断往字符串后加上“0”与“1”（bfs，一个数位一个数位地拓展），判断是否能够整除。

代码

```
# #01426:Find The Multiple
```

```python
#要求用bfs实现
from collections import deque

def bfs(n):
    queue = deque()
    queue.append("1")
    while queue:
        tempo=queue.popleft()
        tempoa=tempo+"0"
        tempob=tempo+"1"
        if int(tempoa)%n==0:
            return tempoa
        if int(tempob)%n==0:
            return tempob
        queue.append(tempoa)
        queue.append(tempob)

while True:
    n=int(input())
    if n==0:
        break
    elif n==1:
        print(1)
    else:
        print(bfs(n))
```

代码运行截图 ==（AC代码截图，至少包含有"Accepted"）==

04115: 鸣人和佐助

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

思路：把查克拉数量也看做一个参数，因为对于同一个方位不同的查克拉数量可能导致不同的选择。随后借助队列来实现bfs，第一次返回的结果即最短路径。

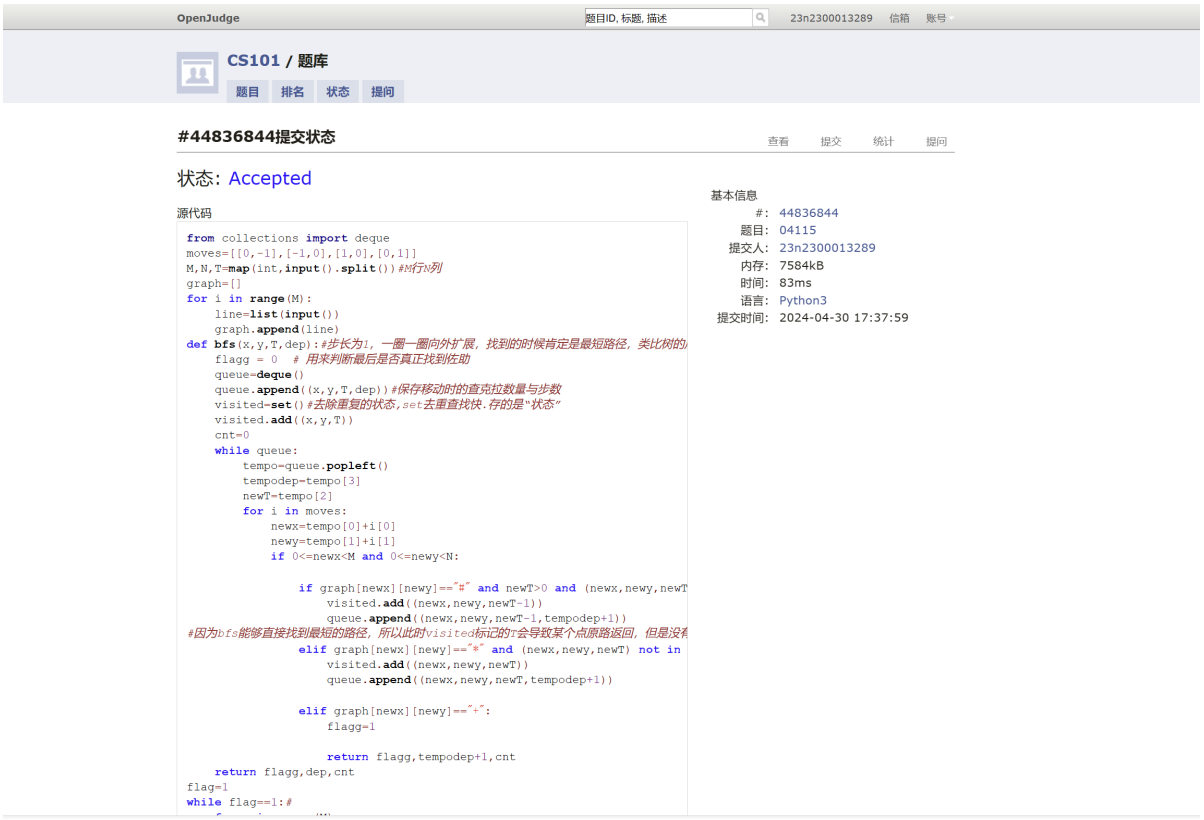
visited去重要储存x，y，方位，队列中要额外储存当前步数

代码

```
# from collections import deque
moves=[[0,-1],[-1,0],[1,0],[0,1]]
M,N,T=map(int,input().split())#M行N列
graph=[]
for i in range(M):
    line=list(input())
    graph.append(line)
def bfs(x,y,T,dep):#步长为1，一圈一圈向外扩展，找到的时候肯定是最短路径，类比树的层次遍历
    flagg = 0  # 用来判断最后是否真正找到佐助
    queue=deque()
    queue.append((x,y,T,dep))#保存移动时的查克拉数量与步数
    visited=set()#去除重复的状态,set去重查找快.存的是“状态”
    visited.add((x,y,T))
    cnt=0
    while queue:
        tempo=queue.popleft()
        tempodep=tempo[3]
        newT=tempo[2]
        for i in moves:
            newx=tempo[0]+i[0]
            newy=tempo[1]+i[1]
            if 0<=newx<M and 0<=newy<N:

                if graph[newx][newy]=="#" and newT>0 and (newx,newy,newT-1) not
in visited:
                    visited.add((newx,newy,newT-1))
                    queue.append((newx,newy,newT-1,tempodep+1))
#因为bfs能够直接找到最短的路径，所以此时visited标记的T会导致某个点原路返回，但是没有关系，因为其他路径一定比返回的路径要更快到达，因为bfs步长为1
                elif graph[newx][newy]=="*" and (newx,newy,newT) not in visited:
                    visited.add((newx,newy,newT))
                    queue.append((newx,newy,newT,tempodep+1))

                elif graph[newx][newy]=="+":
                    flagg=1

                    return flagg,tempodep+1,cnt
    return flagg,dep,cnt
flag=1
while flag==1:#
    for x in range(M):
        for y in range(N):
            if graph[x][y]=="@":
```

```
                ini_x=x
                ini_y=y
                flag=0
                break
flagg,ans,cnt=bfs(ini_x,ini_y,T,0)
if flagg==0:
    print(-1)
else:
    print(ans)
```

代码运行截图 ==（AC代码截图，至少包含有"Accepted"）==



20106: 走山路

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

思路：如何利用好visited来去重是避免MLE的关键。

1.迪杰斯特拉蕴含着动态规划的思想，即visited中储存的每个值都是起点到该点的最小体力消耗。所以一个点只对应一个最优解，visited里只用储存方位即可。

2.何时添加（x，y）到visited里？如果在heappush之后立即添加，可能导致此时的体力消耗并非最优解而被visited锁定。所以应该在最小堆弹出时进行添加，这时候可以保证假如此时有（1，x，y）（2，x，y）.....存在时，弹出最优解。

代码

```
# #20106:走山路
import heapq
m,n,p=map(int,input().split())#m行n列
graph=[]
moves=[[0,-1],[-1,0],[1,0],[0,1]]
for i in range(m):
    line=[x for x in input().split()]
    graph.append(line)

def bfs(ini_x,ini_y):
    flag = 0
    judge=0
    if graph[ini_x][ini_y]=="#" or graph[end_x][end_y]=="#":#题目要求
        return flag,-1
    for i in moves:#判断是否是围城
        if (0<=end_x+i[0]<m and 0<=end_y+i[1]<n) and graph[end_x+i[0]]
[end_y+i[1]]!="#":
            judge=1
            break
    if judge==0:
        return flag,-1

    queue=[]
    heapq.heapify(queue)
    visited=set()
    #因为此时体力值的消耗应该是不严格递增的，题目要求的也是最小体力值消耗即可，所以第一个索引放到
某点已经消耗的体力值，
    # 此时每次步长是当前位置下向周围所要消耗的最小的体力值
    heapq.heappush(queue,(0,ini_x,ini_y))#前一个表示消耗的体力值，0表示第0步

    while queue:
        tempo=heapq.heappop(queue)#不断弹出当前所消耗体力值最小的一步
        visited.add(( tempo[1], tempo [2]))  # 判重标准是来到当前位置时所消耗的体力值,只
保存当前的体力消耗最少节点
        energy = tempo[0]

        if tempo[1]==end_x and tempo[2]==end_y:#要放到前面，避免走最后一步出问题并不一定
是最小体力消耗路径
            flag=1
            return flag,energy
        for i in moves:
            newx=tempo[1]+i[0]
            newy=tempo[2]+i[1]
            if 0<=newx<m and 0<=newy<n:#边界识别
                if graph[newx][newy]!="#":
                    newenergy = energy + abs(int(graph[tempo[1]][tempo[2]])-
int(graph[newx][newy]))
                    if (newx,newy) not in visited:
                        heapq.heappush(queue,(newenergy,newx,newy))

    return flag,-1

for i in range(p):
    ini_x,ini_y,end_x,end_y=map(int,input().split())
    flagg, ans = bfs(ini_x, ini_y)
```

```
    if flagg==1:
        print(ans)
    else:
        print("NO")
```

代码运行截图 ==（AC代码截图，至少包含有"Accepted"）==

05442: 兔子与星空

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

思路：使用了并查集+kruskal的方法，把并查集用类来实现

把所有的边权值，两端点集合起来

然后利用最小堆每次弹出最小权值边开始遍历

期间不断调整并查集判断两端点是否为两兄弟，即与同一个点相联系（用以避免成环），若不是，则把一边接到另一边上。

代码

```
#05442: 兔子与星空
import heapq
class unionandfind:#并查集部分，避免成环
```

```
    def __init__(self,n):
        self.parents=[int(i) for i in range(n)]
        self.height=[0]*n
    def find(self,a):
        if self.parents[a]!=a:
            return self.parents[self.find(self.parents[a])]
        return self.parents[a]
    def union(self,a,b):
        a_fa,b_fa=self.find(a),self.find(b)
        if a_fa != b_fa:
            self.parents[b_fa] = a_fa
            return True
        return False

def krustal(edges,u_and_find):
    ans=0
    while edges:
        tempo = heapq.heappop(edges)
        a = tempo[1]
        b = tempo[2]
        value=tempo[0]
        if u_and_find.union(a,b):
            ans+=value
    return ans

n = int(input())
edges = []
for i in range(n-1):
    line = [x for x in input().split()]
    spot1 = int(ord(line[0])) - 65  # 为了回应并查集中的parents元素位置
    if int(line[1]) != 0:
        for m in range(1,int(line[1])+1):
            spot2 = int(ord(line[2 * m])) - 65
            length = int(line[2 * m + 1])
            heapq.heappush(edges, (length, spot1, spot2))
u_and_find=unionandfind(n)
print(krustal(edges,u_and_find))#
#05442: 兔子与星空
import heapq
class unionandfind:#并查集部分，避免成环
    def __init__(self,n):
        self.parents=[int(i) for i in range(n)]
        self.height=[0]*n
    def find(self,a):
        if self.parents[a]!=a:
            return self.parents[self.parents[a]]
        return self.parents[a]
    def union(self,a,b):
        a_fa,b_fa=self.find(a),self.find(b)
        if a_fa!=b_fa:
            if self.height[a_fa]>self.height[b_fa]:
                self.parents[b_fa]=a_fa
                self.height[a_fa]+=1
            else:
                self.parents[a_fa] = b_fa
                self.height[b_fa] += 1
```

```
            return True
        return False
def krustal(edges,u_and_find):
    ans=0
    queue=set()
    while edges:
        tempo = heapq.heappop(edges)
        a = tempo[1]
        b = tempo[2]
        value=tempo[0]
        if u_and_find.union(a,b):
            if a not in queue:
                queue.add(a)
            if b not in queue:
                queue.add(b)
            ans+=value
    return ans

n = int(input())
edges = []
for i in range(n-1):
    line = [x for x in input().split()]
    spot1 = int(ord(line[0])) - 65  # 为了回应并查集中的parents元素位置
    if int(line[1]) != 0:
        for m in range(1,int(line[1])+1):
            spot2 = int(ord(line[2 * m])) - 65
            length = int(line[2 * m + 1])
            heapq.heappush(edges, (length, spot1, spot2))
u_and_find=unionandfind(n)
print(krustal(edges,u_and_find))
```

代码运行截图 ==（AC代码截图，至少包含有"Accepted"）==

#44837713提交状态

查看 提交 统计 提问

状态: Accepted

基本信息

#: 44837713
题目: 05442
提交人: 23n2300013289
内存: 3692kB
时间: 21ms
语言: Python3
提交时间: 2024-04-30 20:21:22

源代码

```
#05442: 兔子与星空
import heapq
class unionandfind:#并查集部分，避免成环
    def __init__(self,n):
        self.parents=[int(i) for i in range(n)]
        self.height=[0]*n
    def find(self,a):
        if self.parents[a]!=a:
            return self.parents[self.find(self.parents[a])]
        return self.parents[a]
    def union(self,a,b):
        a_fa,b_fa=self.find(a),self.find(b)
        if a_fa != b_fa:
            self.parents[b_fa] = a_fa
            return True
        return False

def krustal(edges,u_and_find):
    ans=0
    while edges:
        tempo = heapq.heappop(edges)
        a = tempo[1]
        b = tempo[2]
        value=tempo[0]
        if u_and_find.union(a,b):
            ans+=value
    return ans

n = int(input())
edges = []
for i in range(n-1):
    line = [x for x in input().split()]
    spot1 = int(ord(line[0])) - 65  # 为了回应并查集中的parents元素位置
    if int(line[1]) != 0:
        for m in range(1,int(line[1])+1):
            spot2 = int(ord(line[2 * m])) - 65
            length = int(line[2 * m + 1])
            heapq.heappush(edges, (length, spot1, spot2))
u_and_find=unionandfind(n)
print(krustal(edges,u_and_find))
```

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。==

感觉后面几道关于算法和bfs的题真的不简单，花了很多时间去理解。

第二题重温树的建构，现在看起来比之前清晰的多

明显感觉图这部分模板性不如树强，尤其是对计概没接触过bfs、dfs的人来说

另外猜测最后一道题感觉数据不太强？并查集部分找父亲没有调用自身函数也能过？？？想知道为啥)

```
class unionandfind:#并查集部分，避免成环
    def __init__(self,n):
        self.parents=[int(i) for i in range(n)]
        self.height=[0]*n
    def find(self,a):
        if self.parents[a]!=a:
            return self.parents[self.parents[a]]
        return self.parents[a]
    def union(self,a,b):
        a_fa,b_fa=self.find(a),self.find(b)
        if a_fa!=b_fa:
            if self.height[a_fa]>self.height[b_fa]:
                self.parents[b_fa]=a_fa
                self.height[a_fa]+=1
            else:
                self.parents[a_fa] = b_fa
                self.height[b_fa] += 1
```

```
            return True
        return False
def krustal(edges,u_and_find):
    ans=0
    queue=set()
    while edges:
        tempo = heapq.heappop(edges)
        a = tempo[1]
        b = tempo[2]
        value=tempo[0]
        if u_and_find.union(a,b):
            if a not in queue:
                queue.add(a)
            if b not in queue:
                queue.add(b)
            ans+=value
    return ans

n = int(input())
edges = []
for i in range(n-1):
    line = [x for x in input().split()]
    spot1 = int(ord(line[0])) - 65  # 为了回应并查集中的parents元素位置
    if int(line[1]) != 0:
        for m in range(1,int(line[1])+1):
            spot2 = int(ord(line[2 * m])) - 65
            length = int(line[2 * m + 1])
            heapq.heappush(edges, (length, spot1, spot2))
u_and_find=unionandfind(n)
print(krustal(edges,u_and_find))
```