Assignment #9: dfs, bfs, & dp

Updated 2107 GMT+8 Nov 19, 2024

2024 fall, Complied by <mark>同学的姓名、院系</mark>

姓名：李彦臻
学号：2300010821
学院：数学科学学院

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。

1. 题目

18160: 最大连通域面积

dfs similar, http://cs101.openjudge.cn/practice/18160

思路：先将繁琐的输入变为格式化的 0/1area 列表，接着使用 dfs 方法遍历每个“岛屿”来计算每个岛屿的面积即可；最后别忘了单独处理“全是水”的情况，因为那样的话 result 列表里会没有数，会报 RE！（这里卡了一会）

代码：

```python

```
```
def max_area(m,n):
    area=[[0]*n for _ in range(m)]#将输入变为格式化的 0/1area 列表
    for i in range(m):
        data=list(map(str,input()))
        for j in range(n):
```

```
            if data[j] == '.':#0 为无棋子（水）&1 为有棋子（陆地）
                area[i][j]=0
            else:area[i][j]=1

    def dfs(x,y):#定义一个函数，用于给定一个岛屿起始点(x,y)时遍历整个岛屿
        length=0#岛屿初始长度
        directions=[(-1,-1),(-1,0),(-1,1),(0,-1),(0,0),(0,1),(1,-1),(1,0),(1,1)]
        if x < 0 or x >= m or y < 0 or y >= n or area[x][y] == 0:
            return length#递归到一个新的点(x,y)时，如果不在定义域内或者是水，则结束，
            #并用 return 返回至上一级函数（不能用 break！！作用完全不一样）
            #（递归函数中 return 的作用：终止当前函数的执行并返回到上一级调用处！）
        else:
            length=1#每一级函数的 length 互相独立，子函数与父函数互不影响
           #（不要 return 1！因为这样虽然会加 1，但也会 return 回上一级函数，直接终止了这一级函数）
            area[x][y]=0#否则，如果是陆地，则标记为 0 表示已经访问过这个点
        for dir in directions:#每个方向挨个儿探索
            length += dfs(x + dir[0],y + dir[1])#对他的一个方向的邻域点继续搜索陆地，并累加 length
            #（如果这个点的邻域全是水，则会全部 return，这个回合就直接结束了）
            #p.s.递归的特点：一个函数可以在自己的定义内部调用自己！

        return length#所有陆地搜索完毕之后，别忘了把最终的总 length 返回为整个函数的函数值！

    result=[]#记录每个岛屿的长度
    for i in range(m):
        for j in range(n):
            if area[i][j] == 1:
                result.append(dfs(i,j))
    if result:#小心全是水的情况。。
        print(max(result))#把整个 area 全部遍历完之后就 OK 啦~
    else:print(0)

n=int(input())
for i in range(n):
    data=list(map(int,input().split()))
    a,b=data[0],data[1]
    max_area(a,b)
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

19930: 寻宝

bfs, http://cs101.openjudge.cn/practice/19930

思路：使用 bfs 方法即可~使用 deque 数据结构来记录当下走到的点的队列，并在每一步中先验证第一个点是否合题，如果不合题再用 popleft 方法删去第一个点并把该点的所有合题的邻域都加入进队列，周而复始，即可做完。

代码：

```python

```

```
from collections import deque

def min_step(m,n):
    maze=[]
    for _ in range(m):
        maze.append(list(map(int,input().split())))

    def bfs(x,y,m,n,maze):#从点(x,y)开始的 bfs 函数
        directions=[(-1,0),(0,-1),(0,1),(1,0)]
      queue=deque([(x,y,0)])#把原 list（第二层括号）里的点（第一层括号）转换到 deque 里面（第三层括号）
```

```
        if maze[x][y] == 2:#小心：别忘了验证第一个点是不是陷阱！！
            return -1

        while queue:#队列里还有可用的点
            a,b,steps=queue.popleft()#将当下可用的点里的最左边的那个点取出来，以它为起点开始移动
            if maze[a][b] == 1:#先验证是否已经到终点了
                return steps#此时的步数就是最短步数，直接 return 即可~
            maze[a][b]=2#否则，如果此刻这个点不是终点，就把它在地图上标为 2，不可再走！
            for dir in directions:
                new_dot=(a+dir[0],b+dir[1],steps+1)#新移动到的点
                    if new_dot[0] in range(m) and new_dot[1] in range(n) and maze[new_dot[0]][new_dot[1]] != 2:
                    #验证新移动到的点是否在地图内、是否未被走过（可以不妨设每个点只被走过一次）
                        queue.append(new_dot)#将这个 ok 的点添加进队列

        return -1#如果 while 循环执行完了还未 return，则说明没有可走路径，返回-1

    k=bfs(0,0,m,n,maze)#不要反复执行这个函数，因为会改变 maze，所以只能执行一次！执行完第一次之后把结果记录下来就好了~
    if k != -1:print(k)
    else:print("NO")

data=list(map(int,input().split()))
a,b=data[0],data[1]
min_step(a,b)
```

代码运行截图 ==（至少包含有"Accepted"）==

04123: 马走日

dfs, http://cs101.openjudge.cn/practice/04123

思路：用 dfs，记录下每个时刻的变量（横坐标、纵坐标、移动步数、此刻已经走过的格子），并对其进行递归即可~
注意：一定别忘了把第一步已经走过的格子加进走过的格子列表里！（寻宝那道题也是因为忘记考虑第一步走的格子是不是陷阱而浪费了不少时间 qwq）

代码：

```python

```

```
def calculate(n,m,x,y):
    walked=[(x,y)]#记录所有走过的格子（别忘了把第一步的格子先放进去！！）
    step=1#这匹马当下走了多少步

    def dfs(x,y,walked,step):#一定要把 walked 也加进 dfs 的变量中！！这样才能在不同的分岔路口有不同的 walked!
        directions=[(-1,-2),(-2,-1),(-1,2),(-2,1),(1,-2),(2,-1),(1,2),(2,1)]
        if step == m*n:#如果此刻已经遍历完毕了，则方法数+1，并结束
            return 1
        count=0#每个子函数中的变量对父函数没有影响~
        for dir in directions:
```

```
            new_x,new_y=x+dir[0],y+dir[1]
            if new_x in range(n) and new_y in range(m) and (new_x,new_y) not in walked:
                #验证这个新点是否在棋盘内且是否未被走过（这里的walked是前一个分叉口遗留的walked！！）
                walked.append((new_x,new_y))#对于这一个新的分岔路口，把这个点添加进walked，不允许再走了
                count += dfs(new_x,new_y,walked,step+1)
                walked.pop()#随后，把这个点重置（移除出walked），从而不影响后续枚举的dir的路口

        return count#全部操作进行完毕之后，别忘了return总方法数！

    print(dfs(x,y,walked,step))

t=int(input())
for i in range(t):
    data=list(map(int,input().split()))
    a,b,c,d=data[0],data[1],data[2],data[3]
    calculate(a,b,c,d)
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

sy316: 矩阵最大权值路径

dfs, https://sunnywhy.com/sfbj/8/1/316

思路：对“矩阵最大权值”那道题的代码稍微修改一下即可~（在外层函数 main（n,m）添加一个变量 max_path 来记录最优路径，当且仅当每次递归到达了终点且这个新的路径的最大值大于此前的最大值时，就更新最大值 max_sum 和最优路径 max_path）

代码：

```python
```

```
def main(n,m):
    matrix = []#规范代码写法（表示定义的等于号两边都有空格，表示并列的逗号前无空格后有空格）
    for _ in range(n):
        matrix.append(list(map(int,input().split())))
    status = [[0]*m for _ in range(n)]#记录每个格子每个时刻的状态
    #（0 表示还未走过，1 表示已经走过）
    status[0][0] = 1
    max_sum = float('-inf')#记录最大值（初始化为负无穷）
    max_path = []#将最优队列初始化为空列表

    def dfs(x, y, now_sum, now_path):#运行到点(x, y)的 dfs 函数
    #p.s.因为 dfs 函数是 main 函数的嵌套函数，因此 main 函数里的变量可以直接使用（不用把 n/m/status/matrix 加入 dfs 函数的变量）
   #需要注意的是,虽然可以使用 main 函数中的变量,但无法修改它!因此,n/m/status/matrix 会被视为 dfs 函数中的局部变量,
    #在每一步 dfs 的每一个 recursion 中对其在 dfs 函数内部进行修改，但不会改变外面的 n/m/status/matrix!
        nonlocal max_sum, max_path
        directions = [(-1,0),(0,-1),(1,0),(0,1)]
        if x == n-1 and y == m-1:
            if now_sum > max_sum:#如果抵达了终点且这个新的路径的最大值大于此前的最大值，就更新最大值
                max_sum = now_sum
                max_path = now_path[:]#同时也更新最优队列；注意，一定不要“max_path = now_path”！
            return
        for dir in directions:
            nx, ny = x+dir[0], y+dir[1]
            if nx in range(n) and ny in range(m) and status[nx][ny] == 0:#验证这个格子是否合题
                status[nx][ny] = 1
                now_path.append((nx,ny))
                dfs(nx, ny, now_sum+matrix[nx][ny], now_path)
```

```
                status[nx][ny] = 0#回溯
                now_path.pop()#回溯
```

dfs(0,0,matrix[0][0],[(0,0)])#最后，执行这个 dfs 函数即可~别忘了初始化 now_sum 和 now_path!

```
    for step in max_path:
        print(f"{step[0]+1} {step[1]+1}")

data = list(map(int,input().split()))
a,b = data[0],data[1]
main(a,b)
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

LeetCode62.不同路径

dp, https://leetcode.cn/problems/unique-paths/

思路：这道题目本身很简单，直接使用 dp 方法即可（建立一个 mxn 表格，把第一行第一列全部初始化为 1，剩下的用递推公式“=左+上”来推即可）；但是 leetcode 的接收数据的方式是真的离谱，研究了半天我才搞明白，原来根本不需要接收数据，leetcode 已经接收好了，只需要把代码的主要部分填充进他的函数模版即可。。

代码：

```python

```

（leetcode 的代码模版只要一复制到 word 里面来就会排版完全错误，因为我直接把我本地的代码复制到这上面来了；本地的代码是按照接受的数据为“m = 3, n = 7”的格式来接收的）

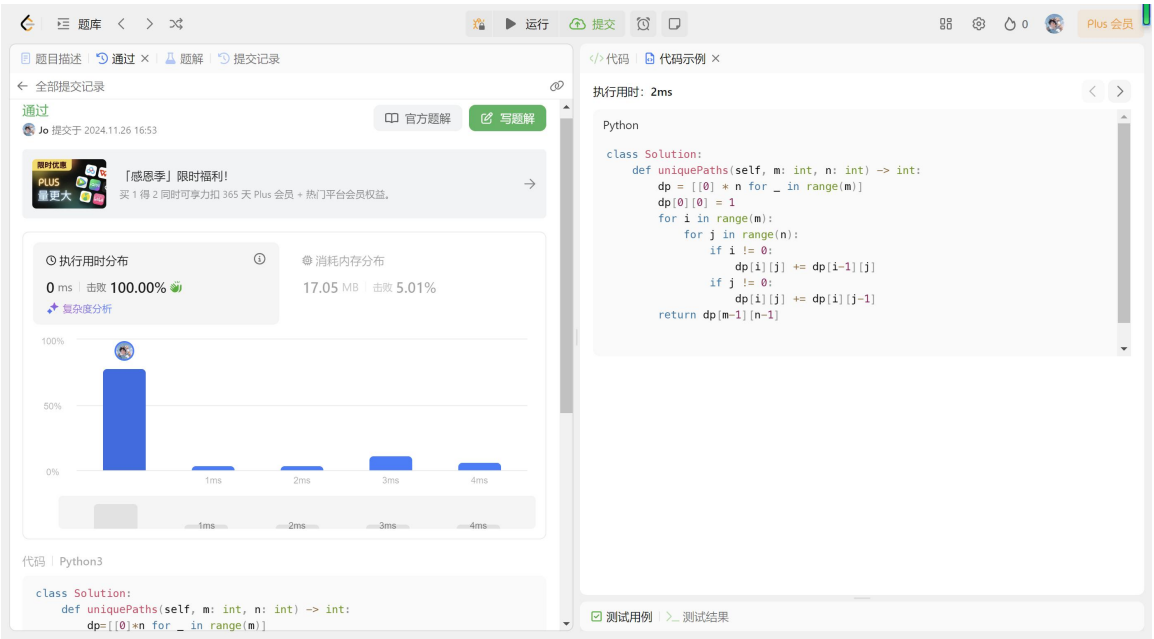
```
data=list(map(str,input().split()))
a=list(map(str,data[2]))[:-1]
b=list(map(str,data[5]))
m=int("".join(map(str,a)))
n=int("".join(map(str,b)))

dp=[[0]*n for _ in range(m)]
for i in range(n):dp[0][i] = 1
for i in range(m):dp[i][0] = 1#先把第一行第一列初始化为 1

for i in range(1,m):
    for j in range(1,n):#对每一行的每一个数递推
        dp[i][j] = dp[i-1][j] + dp[i][j-1]
        #除了第一行 or 第一列以外，每个格子的方法数等于“左+上”

print(dp[m-1][n-1])
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>



sy358: 受到祝福的平方

dfs, dp, https://sunnywhy.com/sfbj/8/3/539

思路：一道比较简单的 dfs 题（暂时没发现他和 dp 有什么关联（？ ）），主体思路就是在每个 dfs 函数里先验证当下的数 m 是否为平方数且不为 0，如果是则直接成立；
如果不是，则不妨设 m 是 k 位数，挨个儿考虑其前 1~k-1 位数的组成的数里是否有平方数且不为 0 的数；如果有是平方数且不为 0 的数，则对 m 除去这个平方数以外的剩下的位数再调用 dfs 函数即可~（如果没有则直接结束）

代码：

```python
```
```
def main(n):
    status = 0

    def dfs(m):
        nonlocal status
        if int(m**0.5) ** 2 == m and m != 0:
            status = 1
            return
        numbers = [digit for digit in str(m)]
        k = len(numbers)
        for i in range(1,k):
            new_num = int("".join(map(str,numbers[:i])))
            if int(new_num**0.5) ** 2 == new_num and new_num != 0:
                rest_num = int("".join(map(str,numbers[i:])))
                dfs(rest_num)

    dfs(n)
    if status == 1:
        print("Yes")
    else:print("No")

n=int(input())
main(n)
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

2. 学习总结和收获

<mark>如果作业题目简单，有否额外练习题目，比如：OJ“计概 2024fall 每日选做”、CF、LeetCode、洛谷等网站题目。</mark>

这次的作业基本全都是 dfs、bfs 类型的题目，因为我之前没怎么联练习过，感觉学起来很困难，尤其是 dfs 那几道题，因为是第一次练习，里面的递归绕的我头晕脑胀，甚至感觉比初学 dp 和 greedy 的时候还痛苦哈哈哈，可能主要是练习的不够多的缘故（多练几道题就好点了）

上来第一题是一个很经典的 dfs 问题，但是我却因为一些小的 bug 卡了好一会（别忘了单独处理“全是水”的情况，因为那样的话 result 列表里会没有数，会报 RE！）

第二题我也犯了一些错误卡住了我好一会，比如第一个格子就是陷阱的情况忘记考虑了（所以我的代码里有一行的注释写着：别忘了验证第一个点是不是陷阱！！），浪费了不少时间；还有就是在验证 bfs 函数结果的时候我引用了 bfs 函数多次，导致出现了问题（反复 WA）：

```
k=bfs(0,0,m,n,maze)#不要反复执行这个函数，因为会改变maze，所以只能执行一次！执行完第一次之后把结果记录下来就好了~
# (像底下这个判断就是麻烦而且不可取的)
if bfs(0,0,m,n,maze) != -1:
    print(bfs(0,0,m,n,maze))
else:print("NO")
```

一开始我没找到问题所在，后来直到某一个案例输出了-1 之后我才意识到我犯了一个多么可笑的错误，下次尽量避免！

在做前几道题的时候，我问了 gpt 一个问题：为什么当我把 dfs 函数放进 main 函数时，不需要在 dfs 函数中再调用一次 main 的变量？gpt 告诉我这是因为 dfs 函数是 main 函数的嵌套函数，因此 main 函数里的变量可以直接使用（不用把 n/m/status/matrix 这些 main 函数的变量加入 dfs 函数的变量）因此，在做第四题的时候，我设置了 max_path 这个在 main 函数中却需要随时根据 dfs 函数的运行结果随时改变的变量的时候，并没有使用 nonlocal 函数（当时我根本不太清楚 nonlocal 这个函数的作用，只知道 global 的作用），而这导致了 max_path 这个变量根本从头到尾没有被改变！

于是，我询问了 gpt，才终于知道了 global、nonlocal、local 的用途，以及全局变量、函数隔离、嵌套函数的之间的关系：

> 你的问题很关键！虽然 `dfs` 是 `main` 的嵌套函数，但这并不意味着嵌套函数中的所有外部变量都可以直接修改。**关键在于作用域和变量的可变性：**
>
> ### Python 的变量作用域规则
>
> 1. **局部变量 (Local)**：
>   - 默认情况下，函数内部定义的变量属于局部作用域，仅在该函数内有效。
> 2. **非局部变量 (Nonlocal)**：
>   - 如果嵌套函数要修改外部函数的局部变量（即外部函数中的变量），需要用 `nonlocal` 声明，否则 Python 会将其视为嵌套函数内的新局部变量。
> 3. **全局变量 (Global)**：
>   - 如果变量在最外层定义，可以通过 `global` 声明在函数中修改全局变量。

简而言之就是，虽然 dfs 函数作为 main 的嵌套函数可以使用 main 函数中的变量，但无法修改它！因此，n/m/status/matrix 这些 main 函数中的变量会被视为 dfs 函数中的局部变量，在每一步 dfs 的每一个 recursion 中对其在 dfs 函数内部进行修改，但不会改变外面的 n/m/status/matrix！（这也符合 recursion 的逻辑）；但是，如果要对 main 函数中的变量在 dfs 中进行修改，就必须使用 nonlocal 函数，来“解禁”！很有收获~

另外，我再做第六题的时候，在判断一个数是否为平方数的时候遇到了问题：

```python
m=100000000000000000001
print(m**0.5)

if m**0.5 is int:
    print(1)

if m**0.5.is_integer():
    print(2)

if m**0.5 % 1 == 0:
    print(3)

import math

if math.sqrt(m) % 1 == 0:
    print(4)

if int(m**0.5) ** 2 == m:
    print(5)
```

我一开始使用的是第一种写法，后来问了 gpt 才知道这个写法大错特错，并在 gpt 的建议下采用了第二种写法，但还是 WA。于是，我自己试验了一下，发现第二种写法无论 m 输入为什么数都会判断 m**0.5 是一个整数（我直到现在都不知道为什么会这样 qwq）；后来，我再问了一下 gpt 才知道，应该使用第三种和第四种写法。

但是后来，为了严谨起见，我又试验了几组数据，经过试验，我发现第四种写法虽然看起来比第三种写法要严谨一点，而且他们俩看起来应该都是对的，但是在 m 是一个充分大且接十分接近平方数的数（如上图）的时候，这两个方法（第三种和第四种方法）都是错的！直到最后，我使用了第五种写法，才终于完全正确～

总结起来就是：第一种方法无论如何都判断 False，第二种无论如何都会判断 True（虽然我不知道为什么 qwq），第三种和第四种看起来比较可靠，但是当 m 是一个非常大的但又离平方数很近的数，例如 100000000000001，就不对！只有第五种方法最可靠~