Assignment #9: dfs, bfs, & dp

Updated 2107 GMT+8 Nov 19, 2024

2024 fall, Complied by 胡杨

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：虽然标的dfs，但是感觉很像讲义例题中的找块的那道题，于是决定写一个bfs（？）

寻找没有在队列中的W，然后将它提取出来，从它开始遍历周围八个方向是W的点，找到一个area+=1，将那个点加到队列中，继续bfs

耗时：30min

bfs模板备份

广度优先搜索 (BFS)一般由**队列**实现,且总是**按层次的顺序进行遍历**，其基本写法如下(可作BFS模板用):

补充：队列

队列（Queue）是一种特殊的线性数据结构，其操作遵循**先进先出**（First In First Out, FIFO）的原则。这意味着最先被添加到队列中的元素也会是最先被移除的元素。队列的基本操作包括：

1. **入队（Enqueue）**：在队列的**尾部**添加一个新的元素。(**append**)
2. **出队（Dequeue）**：从队列的**头部**移除一个元素。(**popleft**)
3. **查看队头元素（Front）**：查看队列中最前面的元素，但不移除它。(**queue[0]**)
4. **查看队尾元素（Rear）**：查看队列中最后面的元素，但不移除它。(**queue[-1]**)
5. **判断队列是否为空（IsEmpty）**：检查队列中是否有任何元素。(**if not queue**)
6. **获取队列大小（Size）**：返回队列中元素的数量。(**len(queue)**)

队列的应用场景

队列在很多计算机科学领域有着广泛的应用，例如：

- **操作系统**：任务调度中使用队列来管理多个进程或线程的执行顺序。
- **网络通信**：在网络数据包传输中，数据包通常会按照接收顺序进行处理。
- **打印队列**：当多台计算机共享一台打印机时，打印请求会被放入一个队列中按顺序处理。
- **广度优先搜索（BFS）**：在图的遍历算法中，队列用于保存待处理的节点，确保所有同一层的节点在进入下一层之前都被处理。

Python 中的队列实现

Python 中可以使用多种方式实现队列，其中一种简单的方法是使用标准库 `collections` 中的 `deque` 类。`deque` 是双端队列的缩写，虽然它支持两端的操作，但也可以用作普通队列：

```
from collections import deque

# 创建一个空队列
queue = deque()

# 入队操作
queue.append('a')
queue.append('b')
queue.append('c')

# 查看队头元素
print(queue[0])  # 输出: 'a'

# 出队操作
item = queue.popleft()
print(item)  # 输出: 'a'

# 判断队列是否为空
if not queue:
    print("队列为空")
else:
    print("队列不为空")

# 获取队列大小
print(len(queue))  # 输出队列中元素的数量
```

在这个例子中，`deque` 被用来创建一个队列，并展示了如何向队列中添加元素（入队）、移除并获取队头元素（出队），以及如何检查队列是否为空和获取队列的大小。

我们使用from collections import deque就满足要求，适用于需要频繁**从队列的两端进行操作**的场景，如广度优先搜索（BFS）、滑动窗口等问题。

> from queue import Queue适用于多线程编程中，需要在多个线程之间安全地共享和传递数据的场景。提供线程安全的特性，内置锁机制，可以在多线程环境中安全地使用。支持阻塞操作，如 `get` 和 `put` 方法可以设置超时时间，等待队列中有数据可用或空间可用。不支持从队列两端进行操作，只能从一端进行插入和删除。

```
from collections import deque
```

```
def bfs(start, end):
    q = deque([(0, start)])  # (step, start)
    in_queue = {start}


    while q:
        step, front = q.popleft() # 取出队首元素
        if front == end:
            return step # 返回需要的结果，如：步长、路径等信息

        # 将 front 的下一层结点中未曾入队的结点全部入队q，并加入集合in_queue设置为已入队
```

下面是对该模板中每一个步骤的说明,请结合代码一起看:

① 定义队列 q，并将起点(0, start)入队，0表示步长目前是0。
② 写一个 while 循环，循环条件是队列q非空。
③ 在 while 循环中，先取出队首元素 front。
④ 将front 的下一层结点中所有**未曾入队**的结点入队，并标记它们的层号为 step 的层号加1，并加入集合in_queue设置为已入队。
⑤ 返回 ② 继续循环。

为了防止走回头路，一般可以设置一个set类型集合**in_queue来记录每个位置是否在BFS中已入过队。**再强调一点，在BFS 中设置的 in_queue 集合的含义是判断结点是否已入过队，而不是**结点是否已被访问。**区别在于：如果设置成是否已被访问，有可能在某个结点正在队列中（但还未访问）时由于其他结点可以到达它而将这个结点再次入队，导致很多结点反复入队，计算量大大增加。因此BFS 中**让每个结点只入队一次**，故需要设置 in_queue 集合的含义为**结点是否已入过队**而非结点是否已被访问。

代码:

```
from collections import deque

dx=[1,1,1,0,0,-1,-1,-1]
dy=[1,0,-1,1,-1,1,0,-1]

def bfs(x,y,n,m,matrix,area=1):
    q=deque()
    q.append((x,y))
    inq=set()
    inq.add((x,y))
    while q:
        cx,cy=q.popleft()
        for i in range(8):
            nx=cx+dx[i]
            ny=cy+dy[i]
            if 0<=nx<n and 0<=ny<m and matrix[nx][ny]=='W' and (nx,ny) not in
inq:
                q.append((nx,ny))
                inq.add((nx,ny))
                area+=1
    return area
```

```
t=int(input())
for _ in range(t):
    n,m=map(int,input().split())
    matrix=[]
    for _ in range(n):
        matrix.append(list(input()))

    max_area=0
    for i in range(n):
        for j in range(m):
            if matrix[i][j]=='W':
                area=bfs(i,j,n,m,matrix)
                max_area=max(max_area,area)
    print(max_area)
```

代码运行截图 （至少包含有"Accepted"）

19930: 寻宝

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

思路：加一圈-1的保护圈，让遍历上下左右四个方向时更加安心（？），设置一个全局变量min_step用于不断更新最短步数。从（1，1）起点开始，寻找下一个可以前进的地方（0或1），将其加入队列，step+1，直到找到为1的点。将min_step与step比较，取最小值。最后执行完程序后的min_step即为最小步数

耗时：20min

代码：

```
from collections import deque

dx=[0,0,1,-1]
```

```
dy=[1,-1,0,0]

def bfs(x,y):
    global min_step

    q=deque()
    q.append((0,(x,y))) #(step,(x,y))
    inq=set()
    inq.add((x,y))

    while q:
        step,(x,y)=q.popleft()
        if graph[x][y]==1:
            min_step=min(min_step,step)
            return

        for i in range(4):
            nx=x+dx[i]
            ny=y+dy[i]
            if graph[nx][ny]!=-1 and graph[nx][ny]!=2 and (nx,ny) not in inq:
                inq.add((nx,ny))
                q.append((step+1,(nx,ny)))

m,n = map(int,input().split())
graph=[[-1]*(n+2)]+[[-1]+list(map(int,input().split()))+[-1] for i in range(m)]+
[[-1]*(n+2)]
min_step=float('inf')

bfs(1,1)
if min_step==float('inf'):
    print('NO')
else:
    print(min_step)
```

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：由于马的移动路线不太规律，需要写一个位移数组dx和dy。dfs访问可以移动到的节点，直到所在位置没有未访问过的节点时返回上一个节点（for循环过完了没有满足if条件的自然就回到上一层去了），如果整个棋盘的节点都被访问过了，count+=1

compile error，加上

```
# pylint: skip-file
```

之后就过了

耗时：30min

代码：

```
# pylint: skip-file
dx=[-2,-2,-1,-1,1,1,2,2]
dy=[-1,1,-2,2,-2,2,-1,1]

def dfs(x,y,dep):
    global count

    if dep==n*m:
        count+=1
        return

    def is_valid(x, y):
        return 0 <= x < n and 0 <= y < m and not visited[x][y]

    for i in range(8):
        nx, ny = x + dx[i], y + dy[i]
        if is_valid(nx, ny):
            visited[nx][ny] = True
            dfs(nx, ny, dep+1)
            visited[nx][ny] = False


t=int(input())
for _ in range(t):
    n,m,x,y=map(int,input().split())
    visited = [[False] * m for _ in range(n)]
    visited[x][y] = True
    count=0
    dep=1
    dfs(x,y,dep)
    print(count)
```

代码运行截图 （至少包含有"Accepted"）

sy316: 矩阵最大权值路径

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

思路：其实做的时候是跟着讲义从迷宫路径数目一道一道做到这里的，所以做本题的时候就基本上是在原来写的模板上改一改就好了。

基本思路是从（0，0）出发，每一次可以走上下左右四个方向，由此建立dx和dy列表。用max_value和max_path分别储存达到最大权值时的数值和路径，用now_value,now_path不断更新当前的权值和路径，每次到达终点时根据now_value和max_value的大小更新权值和路径。对于每走的一步，需要判断其是否合法：不越界，且没有走回头路，然后把原来的now改成nest的状态，持续dfs直到终点

耗时：10min（因为是拿前面的代码改的XD）

代码：

```
def is_valid(x,y):
    return 0<=x<n and 0<=y<m and not visited[x][y]

def dfs(x,y,now_value,now_path):
    global max_value,max_path

    if x==n-1 and y==m-1:
        if now_value>max_value:
            max_value=now_value
            max_path=now_path.copy()
        return

    visited[x][y]=True

    for i in range(4):
        next_x=x+dx[i]
        next_y=y+dy[i]
        if is_valid(next_x,next_y):
            next_value=now_value+maze[next_x][next_y]
```

```
            next_path=now_path+[(next_x,next_y)]
            dfs(next_x,next_y,next_value,next_path)

    visited[x][y]=False

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

visited=[[False]*m for _ in range(n)]

dx=[0,0,1,-1]
dy=[1,-1,0,0]

max_value=float('-inf') #矩阵里的数可能是负的！！
now_value=maze[0][0]
max_path=[]
now_path=[(0,0)]

dfs(0,0,now_value,now_path)
for x,y in max_path:
    print(x+1,y+1)
```

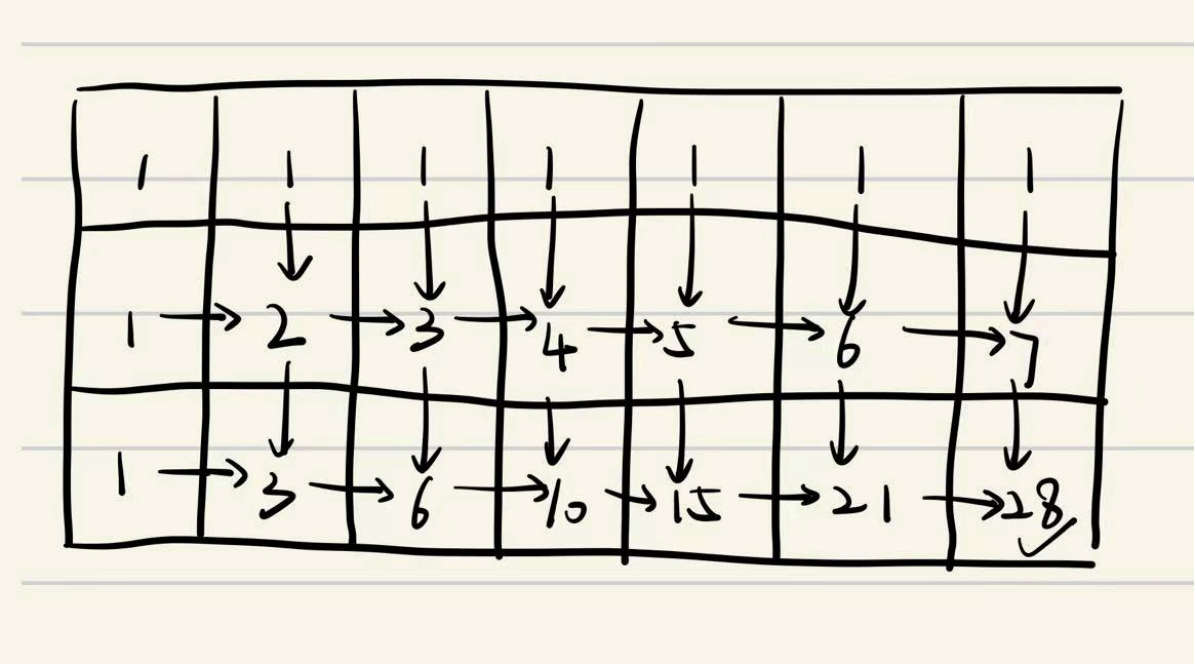
代码运行截图 （至少包含有"Accepted"）

LeetCode62.不同路径

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

思路：这道题的思路和我们生物里的序列比对的打分方法很像，就没有归类它到底是什么背包问题（？）

每一个格子只有两种方法到达，即从它上面一格下来，或者从左边一格过来，故到达该位置的方法就是上面一格和左边一格路径之和，如下图所示：



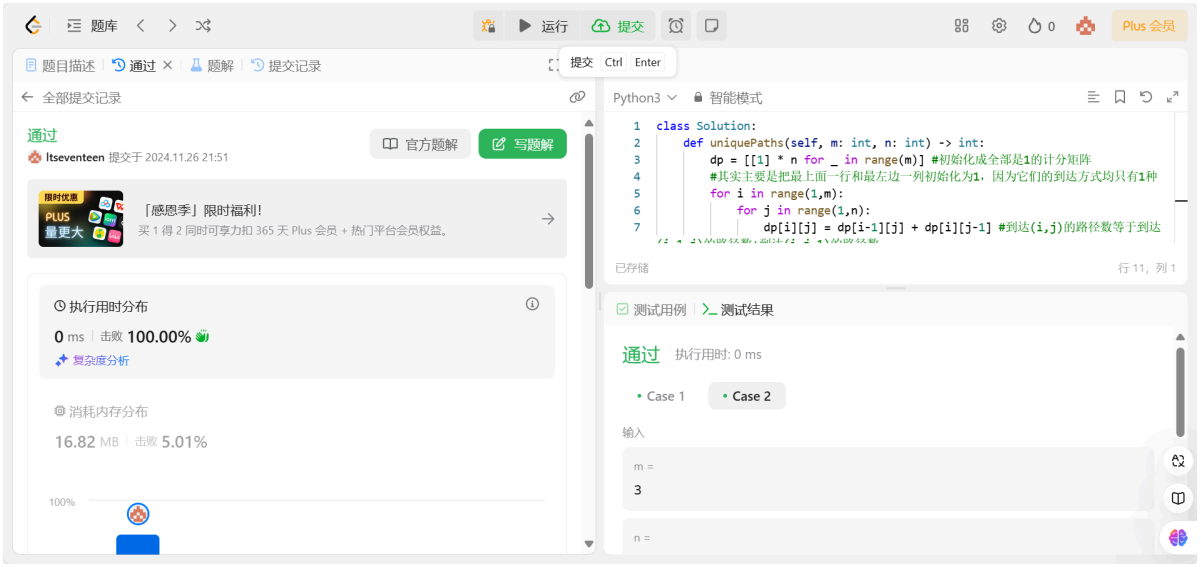
耗时：10min

代码：

```
class Solution:
    def uniquePaths(self, m: int, n: int) -> int:
        dp = [[1] * n for _ in range(m)] #初始化成全部是1的计分矩阵
        #其实主要是把最上面一行和最左边一列初始化为1，因为它们的到达方式均只有1种
        for i in range(1,m):
            for j in range(1,n):
                dp[i][j] = dp[i-1][j] + dp[i][j-1] #到达(i,j)的路径数等于到达(i-1,j)的路径数+到达(i,j-1)的路径数

        return dp[m-1][n-1] #到达(m-1,n-1)的路径数即为结果
```

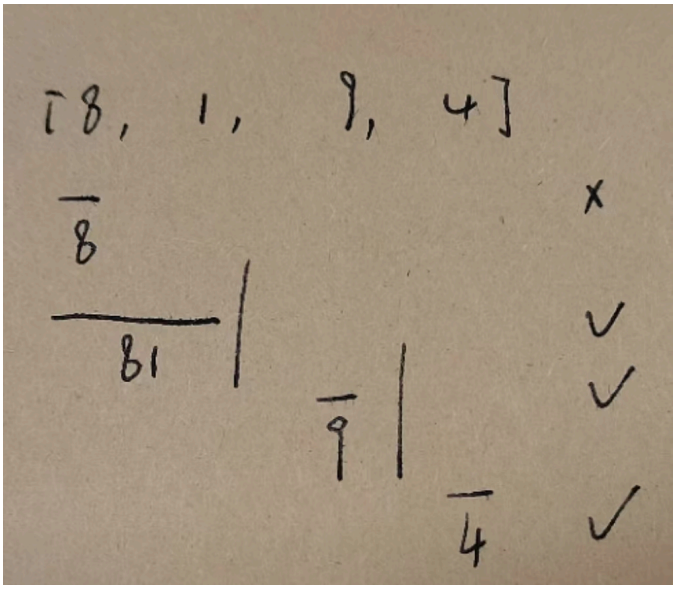
代码运行截图 （至少包含有"Accepted"）



sy358: 受到祝福的平方

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

思路：首先需要把<10**9的平方数都列出来（似乎并没有比枚举更好的办法），然后对于给定的数字id，将其转化成数字的列表，然后从左往右切割看是不是平方数，如果不是就加上下一位看是不是平方数，如果已经是平方数就直接从下一位开始重复上述过程，大概如下图，以8194为例



耗时：30min

代码：

```
#构建平方数列表
squares=[]
for i in range(1,10**5):
    if i**2<=10**9:
        squares.append(i**2)

#平方数判定
```

```python
def is_square(n):
    return n in squares

def is_blessed_id(A):
    num=list(map(int,str(A)))

    def dfs(i):
        if i==len(num):
            return True

        n=0
        for j in range(i,len(num)):
            n=n*10+num[j]
            if is_square(n):
                if dfs(j+1):
                    return True
        return False

    return dfs(0)

A=int(input())
if is_blessed_id(A):
    print("Yes")
else:
    print("No")
```

代码运行截图（至少包含有"Accepted"）

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“计概2024fall每日选做”、CF、LeetCode、洛谷等网站题目。

跟着讲义做完了所有的dfs例题，bfs还没做完，了解了基本模板和队列的基本用法。悲伤地发现好像学了bfs就把dfs忘了（悲）

感觉bfs和dfs的典型例题都有一定的模板，但是我的熟悉程度还有待提升，做题还是会有些吃力

整个作业中竟然觉得不同路径超简单（？），体现了其他领域知识的奇妙的交叉应用（？？）

oj每日选做补天中，本星期ddl超多巨忙，加上高数作业抄都抄不完，感觉身心俱疲，想要休息一下但是一直被任务推着走，甚是痛苦