#!------------------------------------ !#

"""

Name : Tanjim Reza

Student ID : 20101065

Course : CSE422 - Artificial Intelligence

Section : 07

Lab Assignment: 1

"""

#!------------------------------------ !#

""" Task01 Analogy:

1. In Every Step, I need to check the positions for Y and N.

If Y connects with another Y in any direction, then I need to add 1 to the result and start searching from the connected Y.

2. For every DFS, I could have added another MATRIX to check if the position is visited or not,

But changing the position of the Y to N, I could have made the same result without using extra more space.

All the imports, print, pprint statements are for debugging & visualization purpose.

"""

#\*----------------\*#

#\* Global Imports \*#

from pprint import pprint

#\*----------------\*#

###! TASK 01-START !###

#\* Global Variables \*#

max\_count = 0 #\*> Using a Global Variable to store the max count

result = 0

#####? TASK 01: USING DFS ######

#! Creating a 2D Matrix from 'input.txt'

matrix = []

with open('input.txt', 'r') as inputFile:

data = inputFile.readlines() #?> Reading Each Line

for line in data:

matrix.append(line.split()) #?> Adding every line to the matrix

print("\n")

# visited = [[0 for i in range(len(matrix[0]))] for j in range(len(matrix))] UNNEEDED

# pprint(visited)

def search\_neighbour\_positions(matrix, row, col):

"""Search all other possible movements for the current position.

Args:

matrix (list): \_description\_

row (int): row index of the matrix

col (int): column index of the matrix

Returns:

int: max count of the possible movements

"""

global max\_count #\*> Using a Global Variable to store the max count

print("MAX:", max\_count)

max\_count += 1

#!UP

if row-1> 0 and col < len(matrix[0]) and col >= 0 and row-1 >= 0 and \

matrix[row-1][col] == 'Y':

#?> Basic: If the row, col is out of range or the position is N, then already visited or not valid.

matrix[row-1][col] = 'N'

search\_neighbour\_positions(matrix, row-1, col)

#!DOWN

if row+1 < len(matrix) and col < len(matrix[0]) and col >= 0 and row+1 >= 0 and\

matrix[row+1][col] == 'Y':

#?> Basic: If the row, col is out of range or the position is N, then already visited or not valid.

matrix[row+1][col] = 'N'

search\_neighbour\_positions(matrix, row+1, col)

#!LEFT

if col-1 >= 0 and row < len(matrix) and row >= 0 and col-1 < len(matrix[0]) and \

matrix[row][col-1] == 'Y':

#?> Basic: If the row, col is out of range or the position is N, then already visited or not valid.

matrix[row][col-1] = 'N'

search\_neighbour\_positions(matrix, row, col-1)

#!RIGHT

if col+1 < len(matrix[0]) and row < len(matrix) and row >= 0 and \

matrix[row][col+1] == 'Y':

#?> Basic: If the row, col is out of range or the position is N, then already visited or not valid.

matrix[row][col+1] = 'N'

search\_neighbour\_positions(matrix, row, col+1)

#!UP-LEFT

if row-1> 0 and col-1 >= 0 and col-1 < len(matrix[0]) and row-1 < len(matrix) and \

matrix[row-1][col-1] == 'Y':

#?> Basic: If the row, col is out of range or the position is N, then already visited or not valid.

matrix[row-1][col-1] = 'N'

search\_neighbour\_positions(matrix, row-1, col-1)

#!UP-RIGHT

if row-1> 0 and col+1 < len(matrix[0]) and col+1 >= 0 and row-1 < len(matrix) and \

matrix[row-1][col+1] == 'Y':

#?> Basic: If the row, col is out of range or the position is N, then already visited or not valid.

matrix[row-1][col+1] = 'N'

search\_neighbour\_positions(matrix, row-1, col+1)

#!DOWN-LEFT

if row+1 < len(matrix) and col-1 >= 0 and col-1 < len(matrix[0]) and row+1 < len(matrix) and \

matrix[row+1][col-1] == 'Y':

#?> Basic: If the row, col is out of range or the position is N, then already visited or not valid.

matrix[row+1][col-1] = 'N'

search\_neighbour\_positions(matrix, row+1, col-1)

#!DOWN-RIGHT

if row+1 < len(matrix) and col+1 < len(matrix[0]) and col+1 >= 0 and row+1 < len(matrix) and \

matrix[row+1][col+1] == 'Y':

#?> Basic: If the row, col is out of range or the position is N, then already visited or not valid.

matrix[row+1][col+1] = 'N'

search\_neighbour\_positions(matrix, row+1, col+1)

return max\_count

def start\_search(matrix):

global result #\*> Using a Global Variable to store final result

current\_result = 0

global max\_count #\*> Using a Global Variable to store the max count

for row in range(len(matrix)):

for col in range(len(matrix[row])):

# print(row,col) #!Got Each item

if matrix[row][col] == "Y":

current\_result = 0

print("Current Result: ", current\_result)

# result = max(search\_neighbour\_positions(matrix, row, col),result)

current\_result = search\_neighbour\_positions(matrix, row, col)

max\_count = 0

result = max(current\_result, result)

return result-1

task01\_result = start\_search(matrix)

print("Task 01: ", task01\_result)

with open("ouput1.txt", "w") as f:

f.write(str(task01\_result))

f.close()

###! TASK 01-END !###

#! ------------- !#

###! TASK 02: START !###

###! TASK 02: USING BFS !###

matrix = [] #! CLEARING MATRIX so that I can reuse some of my conditions

with open('input2.txt', 'r') as inputFile:

max\_row = int(inputFile.readline())

max\_column = int(inputFile.readline())

data = inputFile.readlines() #?> Reading Each Line

for line in data:

matrix.append(line.split()) #?> Adding every line to the matrix

# print("\n")

# pprint(matrix)

# print("Column:", max\_column)

# print("Row:", max\_row)

visited = [[0 for i in range(int(max\_column))] for j in range(int(max\_row))]

count = 0

allHumans = 0

for row in range(len(matrix)):

for col in range(len(matrix[0])):

if matrix[row][col] == 'H':

allHumans += 1

# print("All Humans:", allHumans)

def AlienVSHuman(matrix):

global allHumans

queue = []

for i in range(int(max\_row)):

for j in range(int(max\_column)):

if matrix[i][j] == 'A':

queue.append([i,j])

while queue:

# print(queue)

row, col = queue.pop(0)

row = int(row)

col = int(col)

# print(row,col)

# print(type(row), type(col))

if col < max\_column and col >= 0 and row-1 < max\_row and row-1 >= 0 and matrix[row-1][col] == 'H':

matrix[row-1][col] = 'N'

queue.append([row-1,col])

visited[row-1][col] = visited[row][col] + 1

allHumans -= 1

if col+1 < max\_column and col+1 >= 0 and row < max\_row and row >= 0 and matrix[row][col+1] == 'H':

matrix[row][col+1] = 'N'

queue.append([row,col+1])

# print(type(row), type(col))

visited[row][col+1] = visited[row][col] + 1

allHumans -= 1

if col-1 < max\_column and col-1 >= 0 and row < max\_row and row >= 0 and matrix[row][col-1] == 'H':

matrix[row][col-1] = 'N'

queue.append([row,col-1])

visited[row][col-1] = visited[row][col] + 1

allHumans -= 1

if row+1 < max\_row and row+1 >= 0 and col < max\_column and col >= 0 and matrix[row+1][col] == 'H':

matrix[row+1][col] = 'N'

queue.append([row+1,col])

visited[row+1][col] = visited[row][col] + 1

allHumans -= 1

max\_value = 0

for i in range(int(max\_row)):

for j in range(int(max\_column)):

if visited[i][j] > max\_value:

max\_value = visited[i][j]

return allHumans, max\_value

output = ""

output += f"Time: {AlienVSHuman(matrix)[1]} minutes\n"

if AlienVSHuman(matrix)[0] == 0:

output += "No one survived"

else:

output += f"{AlienVSHuman(matrix)[0]} survived"

print(output)

with open("output2.txt", "w") as f:

f.write(output)

f.close()

###! TASK 02-END !###

#! ------------- !#