# ===== Task 1 =====

adj\_mat = []

graph = {}

with open('lab1\_q1.txt', 'r') as file:

file = file.readlines()

for line in file:

line = line.split()

adj\_mat.append(line)

def graph\_creator(adj\_mat):

"""Function for creating the dictionary"""

for line\_no, line in enumerate(adj\_mat):

for char\_no, char in enumerate(line):

if char == 'Y':

node = str(line\_no) + str(char\_no)

graph[node] = []

if line\_no - 1 < 0 or char\_no - 1 < 0:

pass

else:

if adj\_mat[line\_no][char\_no] == adj\_mat[line\_no - 1][char\_no - 1]:

graph[node].append(str(line\_no - 1) + str(char\_no - 1))

if line\_no - 1 < 0:

pass

else:

if adj\_mat[line\_no][char\_no] == adj\_mat[line\_no - 1][char\_no]:

graph[node].append(str(line\_no - 1) + str(char\_no))

if line\_no - 1 < 0 or char\_no + 1 >= len(adj\_mat[line\_no]):

pass

else:

if adj\_mat[line\_no][char\_no] == adj\_mat[line\_no - 1][char\_no + 1]:

graph[node].append(str(line\_no - 1) + str(char\_no + 1))

if char\_no + 1 >= len(adj\_mat[line\_no]):

pass

else:

if adj\_mat[line\_no][char\_no] == adj\_mat[line\_no][char\_no + 1]:

graph[node].append(str(line\_no) + str(char\_no + 1))

if line\_no + 1 >= len(adj\_mat) or char\_no + 1 >= len(adj\_mat[line\_no]):

pass

else:

if adj\_mat[line\_no][char\_no] == adj\_mat[line\_no + 1][char\_no + 1]:

graph[node].append(str(line\_no + 1) + str(char\_no + 1))

if line\_no + 1 >= len(adj\_mat):

pass

else:

if adj\_mat[line\_no][char\_no] == adj\_mat[line\_no + 1][char\_no]:

graph[node].append(str(line\_no + 1) + str(char\_no))

if line\_no + 1 >= len(adj\_mat) or char\_no - 1 < 0:

pass

else:

if adj\_mat[line\_no][char\_no] == adj\_mat[line\_no + 1][char\_no - 1]:

graph[node].append(str(line\_no + 1) + str(char\_no - 1))

if char\_no - 1 < 0:

pass

else:

if adj\_mat[line\_no][char\_no] == adj\_mat[line\_no][char\_no - 1]:

graph[node].append(str(line\_no) + str(char\_no - 1))

# W = white if not visited, G = grey if visited, B = black if adjacent nodes explored

color = {}

parent = {}

trav\_time = {} # start, end

output = []

graph\_creator(adj\_mat) # calling the graph creator function

for node in graph.keys():

color[node] = "W"

parent[node] = None

trav\_time[node] = [-1, -1]

time = 0

string\_len = 1

string\_len\_list = []

def dfs\_func(u):

global time, string\_len

color[u] = "G"

trav\_time[u][0] = time # start time at 0th index

time += 1

output.append(u)

if parent[u] == None:

string\_len\_list.append(string\_len) # appending the string length to a list before reseting the string length to 1

string\_len = 1 # reseting string length to 1 if the node doesn't have any parent

for v in graph[u]:

if color[v] == "W":

parent[v] = u

string\_len += 1

dfs\_func(v)

color[u] = "B"

trav\_time[u][1] = time # end time at 1st index

time += 1

# iterating over the dictionary keys which have not been visited yet

for u in graph.keys():

if color[u] == "W":

dfs\_func(u)

string\_len\_list.append(string\_len) # the last string length that was not added to the list

string\_len\_list = string\_len\_list[1:]

string\_len\_list.sort(reverse=True) # sorting the list in reverse order

print(string\_len\_list[0]) # printing the maximum region with Y by printing the 0th index of the sorted list which was in reverse order

# ===== Task 2 =====

adj\_mat = []

with open('lab1\_q2.txt', 'r') as file:

file = file.readlines()

file = file[2:]

for line in file:

line = line.split()

adj\_mat.append(line)

for line in range(len(adj\_mat)):

for char in range(len(adj\_mat[0])):

if adj\_mat[line][char] == 'A':

adj\_mat[line][char] = 2

elif adj\_mat[line][char] == 'H':

adj\_mat[line][char] = 1

else:

adj\_mat[line][char] = 0

def within\_range(adj\_mat, i, j):

"""Function for checking if an index is withing range"""

line = len(adj\_mat)

ch = len(adj\_mat[0])

return i >= 0 and j >= 0 and i < line and j < ch

def time\_passed(adj\_mat):

"""Function for calculating the passed time"""

flag = False

time = 2

line = len(adj\_mat)

ch = len(adj\_mat[0])

while True:

for i in range(line):

for j in range(ch):

if adj\_mat[i][j] == time:

if within\_range(adj\_mat, i + 1, j) and adj\_mat[i + 1][j] == 1:

adj\_mat[i + 1][j] = adj\_mat[i][j] + 1

flag = True

if within\_range(adj\_mat, i - 1, j) and adj\_mat[i - 1][j] == 1:

adj\_mat[i - 1][j] = adj\_mat[i][j] + 1

flag = True

if within\_range(adj\_mat, i, j + 1) and adj\_mat[i][j + 1] == 1:

adj\_mat[i][j + 1] = adj\_mat[i][j] + 1

flag = True

if within\_range(adj\_mat, i, j - 1) and adj\_mat[i][j - 1] == 1:

adj\_mat[i][j - 1] = adj\_mat[i][j] + 1

flag = True

if flag == False:

break

flag = False

time += 1

survivor = 0

for i in range(line):

for j in range(ch):

if adj\_mat[i][j] == 1:

survivor += 1

return survivor, time if time == -1 else time - 2

survivor, time = time\_passed(adj\_mat)

print("Time: ", time)

if survivor > 0:

print(f"{survivor} survived")

else:

print("No one survived")