class Task01:

def tracking\_connections(self, graph\_matrix):

region = 0

for i in range(len(graph\_matrix)):

for j in range(len(graph\_matrix[i])):

if (graph\_matrix[i][j] == 1):

visited\_node = []

region = max(self.dfs(i, j, len(graph\_matrix), len(graph\_matrix[0]), visited\_node, graph\_matrix), region)

return region

def dfs(self, i, j, row, col, visited\_node, graph\_matrix):

visited\_node.append((i, j))

steps = [(-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 1), (1, -1), (1, 0), (1, 1)]

for x, y in steps:

temp\_loc = (i + x, j + y)

if 0 <= temp\_loc[0] < row and 0 <= temp\_loc[1] < col:

if temp\_loc not in visited\_node and graph\_matrix[temp\_loc[0]][temp\_loc[1]]:

self.dfs(temp\_loc[0], temp\_loc[1], row, col, visited\_node, graph\_matrix)

return len(visited\_node)

# ------------driver code-----------#

def driver(self):

input\_text\_files = ['input.txt', 'input 2.txt']

for c, i in enumerate(input\_text\_files):

print('\nTest Case', (c + 1))

with open(i) as f:

lines = f.readlines()

graph\_matrix = []

for j in lines:

row = []

for label in j.split():

if label == 'Y':

row.append(1)

else:

row.append(0)

graph\_matrix.append(row)

max\_region = self.tracking\_connections(graph\_matrix)

print("Maximum Region of Infection", max\_region)

#------------------------------------------------------------------#

class Task02:

def find\_position(self, city):

alien\_positions, human\_positions = [], []

for i in range(len(city)):

for j in range(len(city[i])):

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

alien\_positions.append((i, j))

elif city[i][j] == 'H':

human\_positions.append((i, j))

return alien\_positions, human\_positions

def bfs(self, city, alien):

steps = [(-1, 0), (0, -1), (0, 1), (1, 0)]

time = -1

visited, queue = [], []

for i in alien:

visited.append(i);

queue.append(i)

while queue:

level\_size = len(queue);

while (level\_size):

s = queue.pop(0)

# print(s)

i = s[0];

j = s[1]

for x, y in steps:

temp\_loc = (i + x, j + y)

if 0 <= temp\_loc[0] < len(city) and 0 <= temp\_loc[1] < len(city[0]):

if temp\_loc not in visited and city[temp\_loc[0]][temp\_loc[1]] == 'H':

visited.append(temp\_loc);

queue.append(temp\_loc)

level\_size -= 1

time += 1

return time, len(visited) - len(alien)

# ------------driver code-----------#

def driver(self):

input\_text\_files = ['Question2 input1.txt', 'Question2 input2.txt']

for c, i in enumerate(input\_text\_files):

print('\nTest Case', (c + 1))

with open(i) as f:

lines = f.readlines()[2:]

city = []

for j in lines:

row = []

for label in j.split():

row.append(label)

city.append(row)

# print(city)

alien, human = self.find\_position(city)

print("Positions of Aliens:", alien)

print("Positions of Humans:", human)

time, attacked\_number = self.bfs(city, alien)

print("\nTime:", time, "minutes")

remaining = len(human) - attacked\_number

print(remaining, "survived") if remaining > 0 else print("No one survived\n")

#------------------------------------------------------------------#

print("\nOutput of Question 01")

t1 = Task01()

t1.driver()

print('\n')

print("Output of Question 02")

t2 = Task02()

t2.driver()