A Au bord de la mer

```
n, m, q = map(int, input().split())

carte = []
#Stockage de la carte
for i in range(n):
    carte.append(input())

#Compare si la case aux coordonnées (x,y) est bien du type donné
def compare_cases(x, y, type):
    return carte[x][y] == type

#Définition de la fonction qui vérifie pour une paire de coordonnées
#si la case est en contact avec de l'eau.
def touche_eau(x, y):
    xNord = x - 1
    yEst = y + 1
    xSud = x + 1
    yOuest = y - 1
    eau = "0" #Note: voir au problem B comment généraliser la fonction
    #à un type arbitraire (eau ou terre)

    #Séparation en cas pour la lisibilité:
    # Soi-même
    if compare_cases(x, y, eau):
      return True
    #Nord
    elif (xNord >= 0) and compare_cases(xNord, y, eau): #Opérateur court-circuité
        return True
    #Est
    elif (yEst < m) and compare_cases(x, yEst, eau):
        return True
    #Sud
    elif (xSud < n) and compare_cases(xSud, y, eau):
        return True
    #Ouest
    elif (yOuest >= 0) and compare_cases(x, yOuest, eau):
        return True
    #Ne touche pas d'eau:
    else:
        return False

#Vérification pour chaque coordonnée
for i in range(q):
    x, y = map(int, input().split())
```

```
    if touche_eau(x, y):
        print("YES")
    else:
        print("NO")
```

B Terres isolées

```
n, m = map(int, input().split())

carte = []
#Stockage de la carte
for i in range(n):
    carte.append(input())

#Compare si la case aux coordonnées (x,y) est bien du type donné
def compare_cases(x, y, type):
    return carte[x][y] == type

#Définition de la fonction qui vérifie pour une paire de coordonnées
#si la case est en contact avec de la terre.
#Note : le type à comparer pourait être passé en argument pour généraliser
#la fonction et la rendre utilisable pour les 2 problèmes dans un contexte
#hors de codeforces
def touche_terre(x, y):
    xNord = x - 1
    yEst = y + 1
    xSud = x + 1
    yOuest = y - 1
    typeCase = "#"

    #Séparation en cas pour la lisibilité:
    #Nord
    if (xNord >= 0) and compare_cases(xNord, y, typeCase): #Opérateur court-circuité
        return True
    #Est
    elif (yEst < m) and compare_cases(x, yEst, typeCase):
        return True
    #Sud
    elif (xSud < n) and compare_cases(xSud, y, typeCase):
        return True
    #Ouest
    elif (yOuest >= 0) and compare_cases(x, yOuest, typeCase):
        return True
    #Ne touche pas de terre:
    else:
```

```
        return False

#Compte pour chaque ligne
for x, ligne in enumerate(carte):
    compte = 0
    for y, case in enumerate(ligne):
        #On augmente le compte si la case est une terre et qu'elle n'en touche pas d'autres
        if case == "#" and not touche_terre(x, y):
            compte += 1
    print(compte)
```

C Compter les lacs

Pour compter les lacs, on utilise une stratégie qu'on appelle « Depth-first search », ou « recherche en profondeur ». Cette méthode explore un graphe en allant aussi profondément que possible, puis en remontant progressivement (voir https://fr.wikipedia.org/wiki/Algorithme_de_parcours_en_profondeur). L'idée est de marquer chaque cellule comme visitée ou non-visitée. Lorsqu'on rencontre de l'eau non-visitée, on explore le lac complet et on marque chaque cellule comme visitée. Ainsi, on rencontre chaque lac qu'une seule fois.

```
n,m = map(int,input().split())
c = [input() for _ in range(n)]
# La terre ne fait jamais partie d'un lac
visite = [[(0 if c[i][j] == 'O' else 1) for j in range(m)] for i in range(n)]

# Verifie si un point est dans la carte
def dans_carte(i,j):
    return not (i<0 or i>=n or j<0 or j>=m)

def dfs(a,b):
    stack = [(a, b)]
    while stack:
        i, j = stack.pop()
        if visite[i][j] == 1:
            continue # On a déjà visité ce point
        visite[i][j] = 1
        for x,y in [[1,0],[-1,0],[0,1],[0,-1]]:
            # Si le point n'est pas encore visité, on le visitera!
            if dans_carte(i+x,j+y) and visite[i+x][j+y] == 0:
                stack.append((i+x, j+y))

cnt = 0
for i in range(n):
    for j in range(m):
        if (visite[i][j] == 0):
            dfs(i,j)
```

```
27                 cnt += 1
28 print(cnt)
```