9. Hay un terreno, que podemos pensarlo como una grilla de m filas y n columnas, con trampas y pociones. Queremos llegar de la esquina superior izquierda hasta la inferior derecha, y desde cada casilla sólo podemos movernos a la casilla de la derecha o a la de abajo. Cada casilla i,j tiene un número entero $A_{i,j}$ que nos modificará el nivel de vida sumándonos el número $A_{i,j}$ (si es negativo, nos va a restar $|A_{i,j}|$ de vida). Queremos saber el mínimo nivel de vida con el que debemos comenzar tal que haya un camino posible de modo que en todo momento nuestro nivel de vida sea al menos 1. Por ejemplo, si tenemos la grilla

$$A = \begin{bmatrix} -2 & -3 & 3 \\ -5 & -10 & 1 \\ 10 & 30 & -5 \end{bmatrix}$$

el mínimo nivel de vida con el que podemos comenzar es 7 porque podemos realizar el camino que va todo a la derecha y todo abajo.

---

a) Pensar la idea de un algoritmo de *backtracking* (no hace falta escribirlo).

✓

---

b) Convencerse de que, excepto que estemos en los límites del terreno, la mínima vida necesaria al llegar a la posición i, j es el resultado de restar al mínimo entre la mínima vida necesaria en $i + 1, j$ y aquella en $i, j + 1$, el valor $A_{i,j}$, salvo que eso fuera menor o igual que 0, en cuyo caso sería 1.

✓

---

c) Escribir una formulación recursiva basada en *b*). Explicar su semántica e indicar cuáles serían los parámetros para resolver el problema.

$$\mathrm{TV}_A = \begin{cases} \infty & \text{si } (i > n \wedge j \neq m) \vee (j > m \wedge i \neq n) \\ 0 & \text{si } (i = n+1 \wedge j = m) \vee (i = n \wedge j = m+1) \\ \mathrm{minSig}(i,j) & \text{si } \mathrm{minSig}(i,j) > 0 \\ 1 & \text{sino} \end{cases}$$

$$\mathrm{minSig}(i,j) = \min(\mathrm{TV}_A(i+1,j), \mathrm{TV}_A(i,j+1)) - A_{ij}$$

minSig solo es para que sea menos verboso.

El primer caso devuelve infinito si “se va del tablero”, pero el caso anterior no fué A_{nm}

El segundo caso solo es un caso base para decir que ya pasó A_{nm}

El tercer y cuarto caso son lo explicado en el punto (b)

Asumiendo que es 1-indexed, se resuelve con $\mathrm{TV}_A(1,1)$

---

d) Diseñar un algoritmo de PD y dar su complejidad temporal y espacial auxiliar. Comparar cómo resultaría un enfoque *top-down* con uno *bottom-up*.

Top-down

```
f solve(A, n, m):
```

```
  memo = matriz n*m de valor nulo

  f tv(i,j):
    if (i>n and j!=m) or (j>m and i!=n):
      ret inf

    if (i=n+1 and j=m) or (i=n and j=m+1):
      ret 0

    if memo[i][j] no es nulo:
      ret memo[i][j]

    minSig = min(tv(i+1,j),tv(i,j+1))-A[i][j]

    if minSig > 0:
      memo[i][j] = minSig
    else:
      memo[i][j] = 1

    ret memo[i][j]

  ret tv(0,0)
```

Complejidad espacial: es una matriz de $n \cdot m$ y la pila de recursiones está limitada por la cantidad de estados posibles, que es $n \cdot m$, entonces es $O(n \cdot m)$

Complejidad temporal: también limitada por la cantidad de estados, dado que cada recursión es $O(1)$, nos queda $O(n \cdot m)$

Bottom-up

Nota: lo hice 1-indexed.

```
f solve(A, n, m):

  si n>=m:
    memo = [inf]*(n+1)
    memo[n] = 1

    para cada i en n...1:
      para cada j en m...1:
        memo[i] = max(1,min(memo[i],memo[i+1]) - A[i][j])

  sino:
    memo = [inf]*(m+1)
    memo[m] = 1

    para cada j en m...1:
      para cada i en n...1:
        memo = max(1,min(memo[j],memo[j+1]) - A[i][j])

  ret memo[1]
```

Complejidad espacial auxiliar: es un vector de tamaño $\min(n \cdot m) \Rightarrow O(\min(n \cdot m))$

Complejidad temporal: en ambas opciones posibles son dos for loops anidados de $O(n \cdot m)$

---

e) Dar un algoritmo *bottom-up* cuya complejidad temporal sea $\mathcal{O}(m \cdot n)$ y la espacial auxiliar sea $\mathcal{O}(\text{mín}(m, n))$.

Está en el item anterior.

Implementación:

```
if n < m:
    A = list(zip(*A))
    n, m = m, n

memo = [float('inf')]*(m+1)
memo[m-1] = 1

for i in reversed(range(n)):
    for j in reversed(range(m)):
        memo[j] = max(1, min(memo[j],memo[j+1]) - A[i][j])

return memo[0]
```