Ejercicio 9

Saturday, August 31, 2024 10:50 AM

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.

-2	-3	3
-5	-10	1
10	30	-5

v = 7

v = 5 pos[0][0]

v = 2 pos[0][1]

v = 5 pos[0][2]

v = 6 pos[1][2]

v = 1 pos[2][2]

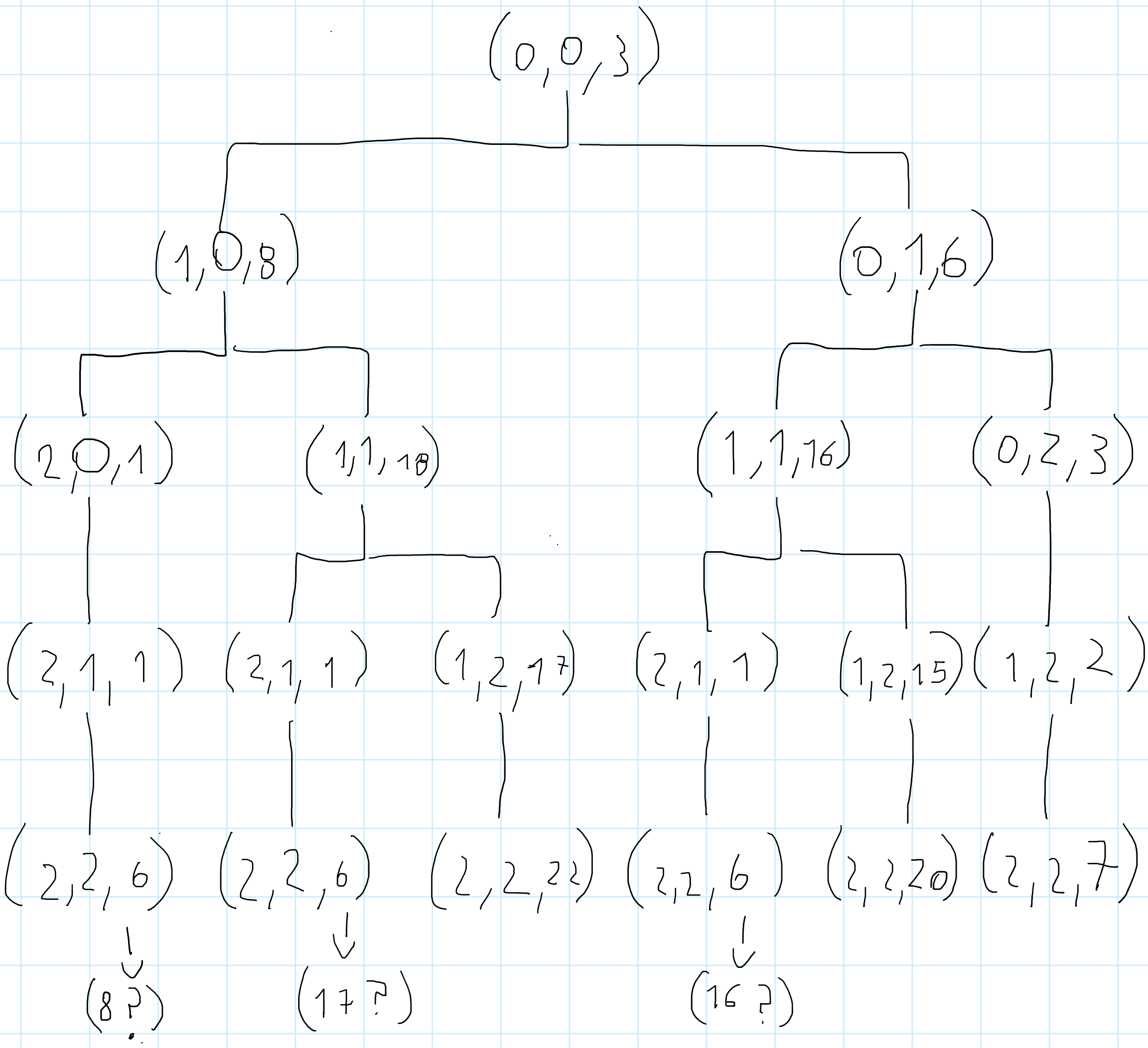
a) Pensar la idea de un algoritmo de *backtracking* (no hace falta escribirlo).

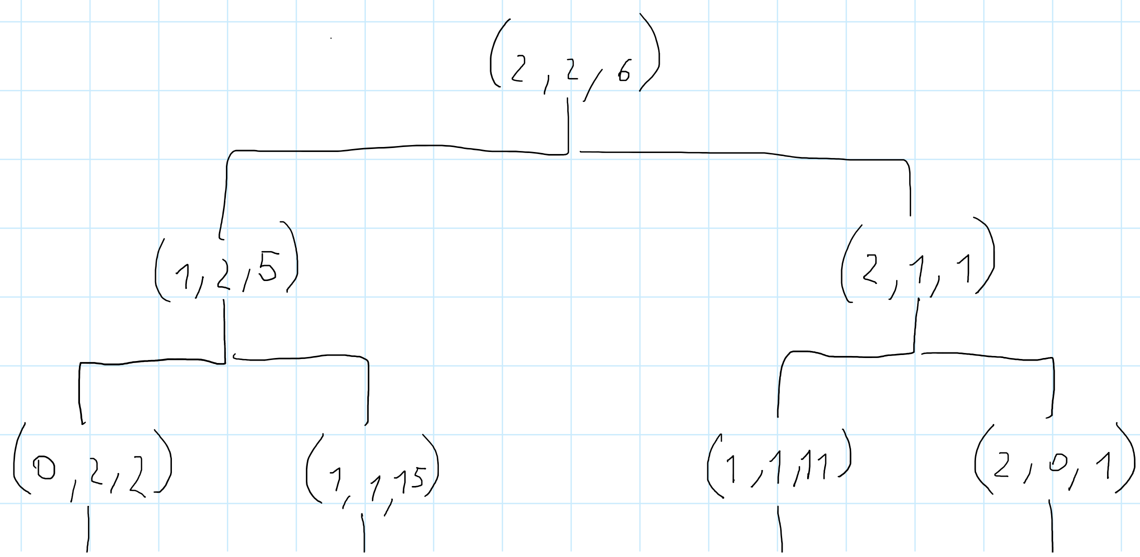
```python
def travesiaVitalBacktracking(mapa, i, j):
    n = len(mapa)
    m = len(mapa[0])

    if i == n-1 and j == m-1:
        return max(1 - mapa[n-1][m-1], 1)

    elif i == n-1:
        return max(travesiaVitalBacktracking(mapa, i, j+1) - mapa[n-1][j], 1)

    elif j == m-1:
        return max(travesiaVitalBacktracking(mapa, i+1, j) - mapa[i][m-1], 1)

    else:
        return max(min(travesiaVitalBacktracking(mapa, i+1, j), travesiaVitalBacktracking(mapa, i, j+1)) - mapa[i][j], 1)
```

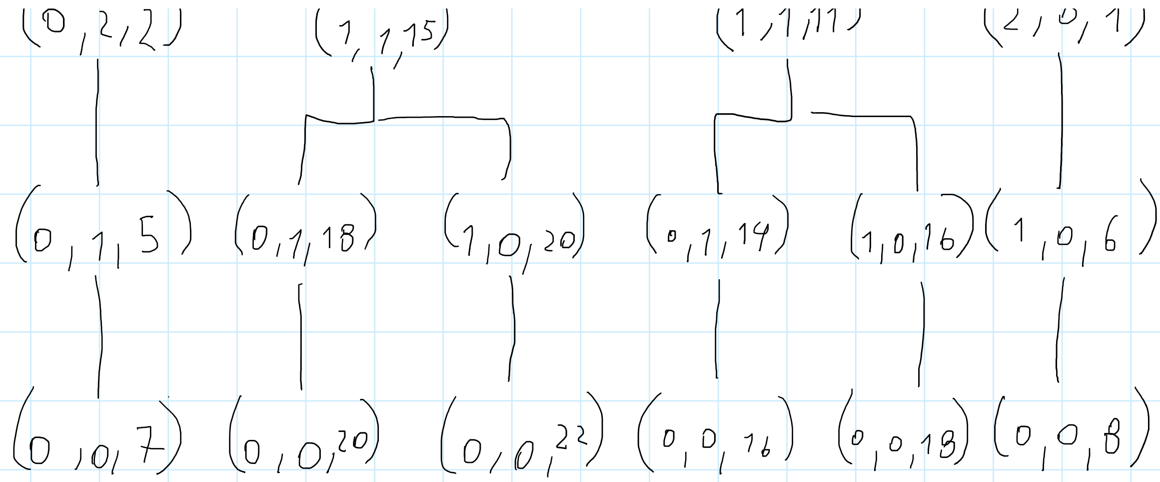
Complejidad temporal $O(2^{(n+m-2)})$

Complejidad espacial $O(n+m)$

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.







$$tv(i,j) = \begin{cases} \max(\min(tv(i,j+1), tv(i-1,j)) - g[i][j], 1) & \text{si } i<n \wedge j<m \\ \max(tv(i,j+1) - g[i][j], 1) & \text{si } i=n \wedge j<m \\ \max(tv(i-1,j) - g[i][j], 1) & \text{si } i<n \wedge j=m \\ \max(1 - g[i][j], 1) & \text{si } i=n \wedge j=m \end{cases}$$

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*.

```python
def travesiaVitalTopDown(mapa, i, j, mem):
    n = len(mapa)
    m = len(mapa[0])

    if mem[i][j] != None:
        return mem[i][j]

    if i == n-1 and j == m-1:
        mem[i][j] = max(1 - mapa[n-1][m-1], 1)

    elif i == n-1:
        mem[i][j] = max(travesiaVitalTopDown(mapa, i, j+1,mem) - mapa[n-1][j], 1)

    elif j == m-1:
        mem[i][j] = max(travesiaVitalTopDown(mapa, i+1, j,mem) - mapa[i][m-1], 1)

    else:
        mem[i][j] = max(min(travesiaVitalTopDown(mapa, i+1, j,mem), travesiaVitalTopDown(mapa, i, j+1,mem)) - mapa[i][j], 1)

    return mem[i][j]
```

Complejidad Temporal O(N.M)

Complejidad Espacial O(N.M)

```python
def travesiaVitalBottomUp(mapa):
    n = len(mapa)
    m = len(mapa[0])

    mem = [[0 for _ in range(m)] for _ in range(n)]

    mem[n-1][m-1] = max(1 - mapa[n-1][m-1], 1)

    for j in range(m-2, -1, -1):
        mem[n-1][j] = max(mem[n-1][j+1] - mapa[n-1][j], 1)

    for i in range(n-2, -1, -1):
        mem[i][m-1] = max(mem[i+1][m-1] - mapa[i][m-1], 1)

    for i in range(n-2, -1, -1):
        for j in range(m-2, -1, -1):
            min_vida_desde_siguiente_paso = min(mem[i+1][j], mem[i][j+1])
            mem[i][j] = max(min_vida_desde_siguiente_paso - mapa[i][j], 1)

    return mem[0][0]
```

Complejidad Espacial O(NM)

Complejidad Temporal O(NM)