

9. Hay un terreno, que podemos pensar como una grilla de m filas y n columnas, con trampas y paciones. 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).

Algoritmo posible
 if estoy en el final
 return valor del final + 1
 if solo puedo ir para abajo ir a la derecha
 return la suma de las casillas hasta el final + 1 por casilla
 else
 return min { recursión bajando + valor casilla actual + 1, recursión a la derecha + valor casilla actual + 1 }

No contemplamos un buen manejo de los resultados y sumas

- 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.

La mínima vida que necesito al llegar a la pos (i,j) es la mínima vida que me va a permitir seguir por un camino norte (abajo o der)

→ Llega a (i,j) con x vida → tiene que cumplir que

$$x + A_{i,j} \geq \min(\text{minVida}(i+1, j), \text{minVida}(i, j+1))$$

$$\rightarrow x \geq \min(\text{minVida}(i+1, j), \text{minVida}(i, j+1)) - A_{i,j}$$

↳ la mínima vida x se da cuando sea igual

Ahora, la parte anterior nos dice que tiene cuenta fuera

↳ $\rightarrow x = 1$. ¿cómo lo haría en lógica?

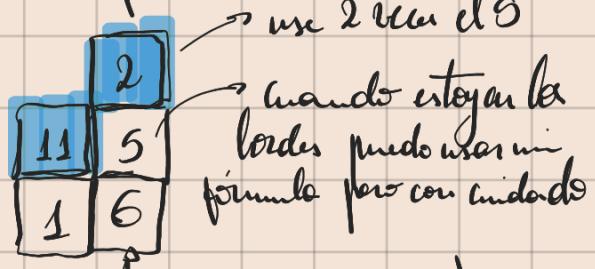
$$x = \max\{1, \underbrace{\min(\text{minVida}(i+1, j), \text{minVida}(i, j+1))}_{\text{para los bordes elijo 1 de los 2}} - A_{i,j}\}$$

$$A = \begin{pmatrix} -2 & -3 & 3 \\ -5 & -10 & 1 \\ 10 & 20 & -5 \end{pmatrix} \rightarrow \text{Caso base: cuando llegue al final}$$

matriz con los mínimos de vida necesaria

$$MV =$$

Si la quiero completar tengo que recalcular cosas que ya calculé



para el caso base
necesito 6 para
al menos tener 1
vida

para los 2 que
reconstruir el 6, me
estoy poniendo

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

$$mV(i, j, A) = \begin{cases} \text{mal } \{ i, -A_{ij+1} \} & \text{para cuando } -A_{ij+1} > 0 \\ \text{mal } \{ i, mV(i, j+1) - A_{ij} \} & \text{para cuando sea negativo} \\ \text{mal } \{ i, mV(i+1, j) - A_{ij} \} & \text{si } i=m \text{ o } j=n \rightarrow \text{borde inferior} \\ \text{mal } \{ i, mV(i+1, j+1) - A_{ij} \} & \text{si } i=m \text{ o } j=n \rightarrow \text{borde derecho} \\ \text{mal } \{ i, \min \{ mV(i+1, j), mV(i, j+1) \} - A_{ij} \} & \text{CC} \end{cases}$$

Para resolver el problema llamo a $mV(0, 0, A)$

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.

① ③ Ya tenemos una f recursiva que resuelve un problema

② Veamos que tiene Suf de subproblemas que ya intrinsecamente tiene

llamadas = Vamos a tener un árbol binario hasta que $i=m \text{ o } j=n$
 Es decir, hasta llegar a un borde. Va a ser un árbol completo hasta ese momento. $\rightarrow \Omega(2^{\min(n,m)})$

sustituciones: $O(mn)$ \rightarrow dentro de una m filas tengo n columnas
 para displayarme a través

$$m \cdot n \ll 2^{\min(n,m)} \left(\text{Si } \min(n,m) = n \right) \leftrightarrow m \ll 2^n/n \\ \left(\text{Si } \min(n,m) = m \right) \leftrightarrow n \ll 2^m/m$$

En estos dos casos hay suf de subproblemas.

Algorithm for PD

$mV(i, j)$

if $i=m$ and $j=n$ then

return $\max\{1, -A_{ij} + 1\}$

else

if $i=m$ and $j \neq n$ then

return $\max\{1, mV(i, j+1) - A_{ij}\}$

else if $i \neq m$ and $j=n$

return $\max\{1, mV(i+1, j) - A_{ij}\}$

else

: return $\max\{1, \min\{mV(i+1, j), mV(i, j+1)\} - A_{ij}\}$

endif

endif

PD

Definir una matriz M de $m \times n$ inicializada en $0 \leq \text{val} \geq 1$

$\text{uv}(i, j)$

if $i = m \wedge j = n$

return $\max\{1, -A_{ij} + 1\}$

else if $i = m \wedge j \neq n$

if $M[i][j] = 0$ then

$M[i][j] = \max\{1, \text{uv}(i, j+1) - A_{ij}\}$

endif

return $M[i][j]$

else if $i \neq m \wedge j = n$ then

if $M[i][j] = 0$

$M[i][j] = \max\{1, \text{uv}(i+1, j) - A_{ij}\}$

endif

return $M[i][j]$

else

if $M[i][j] = 0$ then

$M[i][j] = \max\{1, \min\{\text{uv}(i+1, j), \text{uv}(i, j+1)\} - A_{ij}\}$

endif

return $M[i][j]$

endif

Complejidad T: $O(mn)$ E: $\Theta(mn)$

El enfoque TD nos permite descomponer los cálculos para saber que queremos reconstruir las soluciones y qué datos

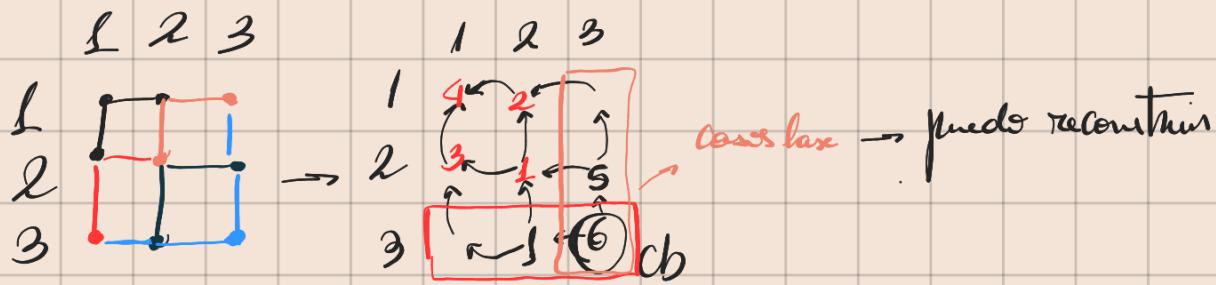
El enfoque BV nos permite optimizar la complejidad espacial a costa de memoria. Se pierde la posibilidad de reconstrucción.

A veces incluso con TD lo calculamos todos los subproblemas, solo los necesarios. Con BV no calculamos TODOS

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

Caso del gráfico. Véase como se llena un matriz

$$mV(i, j, A) = \begin{cases} \max\{1, -A_{ij} + 1\} & i = m \text{ o } j = n \\ \max\{1, mV(i, j+1) - A_{ij}\} & i = m \text{ o } j = n \\ \max\{1, mV(i+1, j) - A_{ij}\} & i = m \text{ o } j = n \\ \max\{1, \min\{mV(i+1, j), mV(i, j+1)\} - A_{ij}\} & \text{C.C.} \end{cases}$$



→ vamos a ir subiendo la fila y dejando fija la columna

PD - BU

$mV(i, j)$

vector M de tamaño m

vector N de tamaño n

$M[m-1] = \max\{1, -A_{m,n} + 1\}$

$N[n-1] = M[m-1]$

for ($i = n-2, i \geq 0, i--$) {

$N[i] = \max\{1, N[i+1] - A_{i,n}\}$

}

for ($i = m-2, i \geq 0, i--$) {

$M[i] = \max\{1, M[i+1] - A_{i,n}\}$

}

for ($i = m-2, i \geq 0, i--$) {

for ($j = n-1, j \geq 0, j--$) {

if $j = n-1$

$N[j] \leftarrow M[i]$

else

$N[j] \leftarrow \max\{1, \min\{N[j], N[j-1]\} - A_{ij}\}$

endif

}

return $N[0]$

Complejidad $\mathcal{T}(O(m \cdot n))$ y $E \cdot O(m \cdot n) = O(\max(m, n))$

para el min me pide

fixar cual es menor y hacerlo

por filas o columnas

Vamos a hacerlo con complejidad espacial $\min(m, n)$

PD - BU

elijo el menor número entre # filas y # columnas $\rightarrow k = \min(m, n)$

vector M de tamaño k

vector N de tamaño k en 0

$M[k-1] = \max\{1, -A_{mn} + 1\}$

for($i=m-2, i \geq 0, i--$) { // hacer este proceso de arriba abajo que sea menor

 for($j=n-1, j \geq 0, j--$) {

 if $j = n-1$

$N[j] \leftarrow M[i]$

 else

$N[j] \leftarrow \max\{1, \min\{N[j], N[j+1]\} - A_{ij}\}$

 endif

}

swaf(N, M)

return $N[0]$