ESERCIZIO 1.

Considerate una griglia $n \times m$ dove la posizione in basso a sinistra sia identificata dalle coordinate $(0,0)$ e la posizione in alto a destra sia identificata dalle coordinate $(n-1, m-1)$.

Un robot è inizialmente nella posizione $(0,0)$ e da una generica posizione (a, b) può muoversi nelle posizioni, $(a, b+1)$, $(a+1, b)$ e $(a+1, b+2)$.

Trovate un algoritmo che, dati i valori di n e m, conti in quanti modi diversi il robot può raggiungere le coordinate (n-1, m-1).

Ad esempio:

per $n = m = 3$ la risposta deve essere 8. Abbiamo infatti i seguenti possibili cammini:

$$\begin{bmatrix} 2 & 3 & 4 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 3 & 4 \\ 1 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 4 \\ 1 & 2 & 3 \\ 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 2 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 3 & 4 \\ 0 & 2 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 4 \\ 0 & 2 & 3 \\ 0 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 4 \\ 0 & 0 & 3 \\ 0 & 1 & 2 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 & 2 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

Progettare un algoritmo che risolve il problema in tempo $\Theta(n \cdot m)$.

Motivare BENE la correttezza e la complessità dell'algoritmo proposto.

Utilizziamo una tabella di dimensioni $n \times m$.

$T[i][j] =$ il numero di cammini che dalla cella $(0,0)$ arrivano alla cella (i, j) .

La soluzione al problema sarà $T[n-1][m-1]$.

Resta solo da definire la ricorrenza per il calcolo delle $\Theta(n \cdot m)$ celle della tabella .

La formula ricorsiva che permette di ricavare $T[i][j]$ dalle celle precedentemente calcolate è la seguente:

$$T[i][j] = \begin{cases} 1 & \text{se } i = 0 \text{ or } j = 0 \\ T[i-1][j] + 1 & \text{se } j = 1 \\ T[i-1][j] + T[i][j-1] + T[i-1][j-2] & \text{altrimenti} \end{cases}$$

La ricorrenza è corretta per i seguenti motivi:

- per $i = 0$ si ha $T[0][j] = 1$ perché posso raggiungere la cella a partire da $(0,0)$ con un unico cammino (quello che effettua sempre spostamenti verso destra)
- per $j = 0$ si ha $T[i][0] = 1$ perché posso raggiungere la cella a partire da $(i,0)$ con un unico cammino (quello che effettua sempre spostamenti verso l'alto)
- per $j = 1$ e $i > 0$ si ha $T[i][j] = T[i-1][j] + T[i][j-1]$ perché posso raggiungere la cella (i, j) sia con cammini che provengono da $(i-1,j)$ che con cammini che provengono da $(i, j-1)$. Ma $T[i][j-1] = T[i][0] = 1$. Si ha quindi $T[i][j] = T[i-1][j] + 1$
- In tutti gli altri casi si ha $T[i][j] = T[i-1][j] + T[i-1][j] + T[i-1][j-2]$.

Implementazione:

```
def es1(n,m):
    T=[[[0]  for _ in range(m)] for _ in range(n)]
    for i in range(n):
        for j in range(m):
            if i==0 or j==0: T[i][j]=1
            elif j==1: T[i][1]=T[i-1][j]+1
            else: T[i][j]=T[i-1][j]+T[i][j-1]+T[i-1][j-2]
    return T[n-1][m-1]



>>> es1(1,1)
1
>>> es1(2,2)
2
>>> es1(2,3)
4
>>> es1(3,2)
3
```

Complessità $\Theta(n \cdot m)$

ESERCIZIO 2

Data una stringa binaria$(b_0, b_1, \ldots, b_{n-1})$, definiamo il vantaggio alla posizione i come la differenza tra il numero di 1 e il numero di 0 nelle posizioni $(0, \ldots, i)$. Ad esempio per la stringa 010010111000 il vantaggio nelle dodici posizioni è rispettivamente:

$$-1, 0, -1, -2, -1, -2, -1, 0, 1, 0, -1, -2.$$

Dati tre numeri n, a e b, con $a \leq 0 < b$, definiamo $S(n, a, b)$ l'insieme di stringhe binarie di lunghezza n il cui vantaggio in ogni posizione sia compreso tra a e b inclusi. Ad esempio:

$S(3, -1,1) = \{010, 011, 100, 101\}$ e $S(4,0,3) = \{1010, 1011, 1100, 1101, 1110\}$.

Trovate un algoritmo che dati n, a e b, stampi tutte le stringhe in $S(n, a, b)$.

La complessità dell'algoritmo deve essere $O(n \cdot S(n, a, b))$.

Motivare BENE la correttezza e la complessità dell'algoritmo proposto

Algoritmo:

- Utilizziamo un algoritmo di backtracking per la stampa di tutte le stringhe binarie sol di lunghezza n con l'aggiunta di funzioni di taglio.
- Utiliziamo: una variabile $vantaggio$ che riporta il vantaggio della stringa fin'ora costruita vale a dire la differenza tra il numero di uni e il numero di zeri inseriti fino a questo momento nella soluzione che si sta costruendo.
- al passo i dobbiamo fare attenzione a mantenere il vantaggio entro i limiti, vale a dire $a \leq vantaggio \leq b$, quindi:
  - nella posizione $sol[i]$ inserisco 0 e decremento $vantaggio$ se e solo se $vantaggio > a$
  - nella posizione $sol[i]$ inserisco 1 e incremento $vantaggio$ se e solo se $vantaggio < b$

Implementazione

```python
def es2(n,a,b):
    sol=[0]*n
    es2b(0, n, a, b, 0, sol)



def es2b(i, n, a, b, vantaggio, sol):
    if i==n:
        print(''.join(sol))
        return
    if vantaggio > a:
        sol[i]='0'
        es2b(i+1, n, a, b, vantaggio-1, sol)
    if vantaggio < b:
        sol[i]='1'
        es2b(i+1, n, a, b, vantaggio+1, sol)
```

- Nota che nell'albero di ricorsione prodotto dall'esecuzione di $es2$ un nodo viene generato solo se porta ad una foglia da stampare.
- Possiamo quindi dire che la complessità dell'esecuzione di $es2(n, a, b)$ richiederà tempo

$$O(S(n, a, b) \cdot h \cdot f(n) + S(n, a, b) \cdot g(n))$$

  dove:

  - $h = n$ è l'altezza dell'albero.
  - $f(n) = \Theta(1)$ è il lavoro di un nodo interno.
  - $g(n) = \Theta(n)$ è il lavoro di una foglia
- Quindi il costo di $es2(n, a, b)$ è $\Theta(nS(n, a, b))$.