ESERCIZIO 1.

Una pedina è posizionata sulla casella $(0,0)$ in alto a sinistra di una scacchiera $n \times n$ e mediante una sequenza di mosse tra caselle adiacenti deve raggiungere la casella $(n-1,n-1)$ in basso a destra. Una pedina posizionata sulla casella (i,j) ha al più due mosse possibili: spostarsi verso La sequenza di caselle toccate dalla pedina nello spostarsi da $(0,0)$ a $(n-1,n-1)$ determina un cammino. Ogni casella della della scacchiera è labellata con 0 o 1. Un cammino è definito lecito se non contiene caselle adiacenti con la stessa label.

Descrivere un algoritmo che data una matrice binaria M di dimensioni $n \times n$ calcola in tempo $O(n^2)$il numero di cammini leciti di$M$.

Ad esempio per la matrice $M = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$ la risposta è 3 (i cammini leciti

sono infatti: $\begin{bmatrix} 1 & 0 & 1 \\ - & - & 0 \\ - & - & 1 \end{bmatrix}$, $\begin{bmatrix} 1 & 0 & - \\ - & 1 & 0 \\ - & - & 1 \end{bmatrix}$, $\begin{bmatrix} 1 & - & - \\ 0 & 1 & 0 \\ - & - & 1 \end{bmatrix}$).

Progettare un algoritmo che risolve il problema in tempo $O(n^2)$.

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

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

$T[i, j] =$ **lunghezza massima per cammino lecito che termina in posizione** (i, j)

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

Resta solo da definire la ricorrenza per il calcolo delle $\Theta(n^2)$ celle

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

$$T[i,j] = \begin{cases} 1 & \text{se } i = j = 0 \\ T[0, j-1] & \text{se } i = 0 \text{ e } M[0, j-1] \neq M[0,j] \\ 0 & \text{se } i = 0 \text{ e } M[0, j-1] = M[0,j] \\ T[i-1, 0] & \text{se } j = 0 \text{ e } M[i-1, 0] \neq M[i,0] \\ 0 & \text{se } j = 0 \text{ e } M[i-1, 0] = M[i,0] \\ T[i-1,j] + T[i,j-1] & \text{se } M[i-1,j] \neq M[i,j] \text{ and } M[i,j-1] \neq M[i,j] \\ T[i,j-1] & \text{se } M[i-1,j] = M[i,j] \text{ and } M[i,j-1] \neq M[i,j] \\ T[i-1,j] & \text{se } M[i-1,j] \neq M[i,j] \text{ and } M[i,j-1] = M[i,j] \\ 0 & \text{se } M[i-1,j] = M[i,j] \text{ and } M[i,j-1] = M[i,j] \end{cases}$$

La ricorrenza è corretta per i seguenti motivi:

- esiste un solo cammino lecito che termina in $(0,0)$
- se $i = 0$ esiste potenzialmente un solo cammino lecito che termina in j e proviene da $j - 1$
- se $j = 0$ esiste potenzialmente un solo cammino lecito che termina in i e proviene da $i - 1$
- se un cammino lecito che arriva a (i, j) può provenire da $(i - 1, j)$ se $M[i - 1, j] \neq M[i, j]$ o anche da $(i, j - 1)$se $M[i, j - 1] \neq M[i, j]$. Devo quindi considerare i 4 casi possibili.

```python
def es1(M):
    n=len(M[0])
    T=[[0 for i in range(n)] for j in range(n)]
    T[0][0]=1
    for i in range(n):
        for j in range(n):
            if j and M[i][j]!=M[i][j-1]:
                T[i][j]+=T[i][j-1]
            if i and M[i][j]!=M[i-1][j]:
                T[i][j]+=T[i-1][j]
    return  T[n-1][n-1]
```

Complessità $\Theta(n^2)$

```python
>>> M=[[1,0,1],[0,1,0],[0,1,1]]
>>> es1(M)
3
>>> M1=[[1,1,0],[0,0,0],[1,0,1]]
>>> es1(M1)
1
```

ESERCIZIO 2

Progettare un algoritmo che data una stringa X lunga n sull'alfabeto $\{0,1,2\}$ stampa tutte le stringhe lunghe n sull'alfabeto $\{0,1,2\}$ che concordano con la stringa X in esattamente una posizione.

Ad esempio per $X =' 200'$ l'algoritmo deve stampare, non necessariamente nello stesso ordine, le seguenti 12 stringhe:

$$110 \quad 101 \quad 102 \quad 120 \quad 010 \quad 001 \quad 002 \quad 020 \quad 211 \quad 212 \quad 221 \quad 222$$

L'algoritmo proposto deve avere complessità $O(nS(n))$ dove $S(n)$ è il numero di stringhe da stampare.

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

Algoritmo:

- Utilizziamo un algoritmo di backtracking per la stampa di tutte le stringhe ternarie *sol* lunghe n con l'aggiunta di funzioni di taglio.
- Utilizzo un flag p che vale 0 se nella stringa *sol* che sto costruendo non c'è ancora una posizione in cui un simbolo concorda con quello della stringa X
- al passo i:
  - se $p = 1$ allora devo aggiungere necessariamente un simbolo diverso da $X[i]$
  - se $p = 0$ e $i = n-1$ allora devo necessariamente aggiungere il simbolo $X[i]$
  - se $p = 0$ e $i < n - 1$ posso aggiungere un qualsiasi simbolo ma se aggiungo il simbolo $X[i]$ devo porre $p = 1$

Implementazione:

```python
def BT(X,i=0,p=0,sol=[]):
    if i==len(X):
        print (''.join(sol))
    else:
        if p==1:
            I={'0','1','2'}-{X[i]}
            for x in I:
                sol.append(x)
                BT(X,i+1,1,sol)
                sol.pop()
        else:
            if i==len(X)-1:
                sol.append(X[i])
                BT(X,i+1,1,sol)
                sol.pop()
            else:
                I={'0','1','2'}
                for x in I:
                    sol.append(x)
                    if x!=X[i]:
                        BT(X,i+1,0,sol)
                    else:
                        BT(X,i+1,1,sol)
                    sol.pop()
```

```
>>> X='200'
>>> BT(X)
110
101
102
120
010
001
002
020
211
212
221
222
```

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

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

dove:

  - $h = n$ è l'altezza dell'albero.
  - $f(n) = O(1)$ è il lavoro di un nodo interno.
  - $g(n) = O(n)$ è il lavoro di una foglia

- Quindi il costo di $BT(X)$ è $O(nS(n))$.

ESERCIZIO 3

In generale un grafo pesato G può avere diversi alberi di copertura di peso minimo. Dimostrare o confutare che se i pesi degli archi di G sono tutti distinti allora G ha un unico albero di copertura.

L'affermazione è vera! Una possibile prova è per assurdo.
Assumiamo per assurdo che G abbia due diversi alberi di copertura di peso minimo T_1 e T_2. I due alberi hanno dunque un certo numero di archi distinti, tra tutti gli archi che appartengono ad uno solo degli alberi sia e quello di costo minimo e senza perdere di generalità assumiamo appartenga all'albero T_1 (in caso contrario il ragionamento è completamente simmetrico). Posso inserire l'arco e in T_2 e formare così un ciclo C. Non tutti gli archi di questo ciclo sono in T_1 (in caso contrario il ciclo sarebbe presente anche nell'albero T_1) sia dunque e' un arco di C non presente in T_1. Questo è dunque un arco presente unicamente in T_2 ed ha un costo superiore a quello di e (per come è stato scelto e). Eliminiamo dunque l'arco e' da T_2 ottenendo così un nuovo albero di copertura T'. Nota che questo nuovo albero di copertura T' ha costo inferiore all'albero T_2 (abbiamo infatti sostituito in T_2 e' con l'arco e di peso inferiore). Deduciamo quindi che T_2 non poteva essere un minimo albero di copertura ottenendo l'assurdo.