ESERCIZIO 1

Il display di un telefonino si presenta come di seguito indicato:

1	2	3
4	5	6
7	8	9
*	0	#

Cerchiamo un particolare numero telefonico e sappiamo che:

- il numero è composto da n cifre.
- non contiene cifre uguali adiacenti
- nel comporre il numero sul tastierino basta spostarsi solo tra tasti adiacenti in orizzontale o verticale

Ad esempio, per $n = 7$, la combinazione 12108586996 non è di certo il numero telefonico che cerchiamo a causa della presenza delle seguenti tre coppie di cifre adiacenti 10 e 86 e 99.

Progettare un algoritmo che, dato n, restituisce il numero di combinazioni possibili per il numero telefonico da ricercare.
Ad esempio:

- per $n = 1$ la risposta dell'algoritmo deve essere 10
- per $n = 2$ la risposta dell'algoritmo deve essere 26.
  (i numeri possibili sono infatti: $08, 12, 14, 21, 23, 25, 32, 36, 41, 45, 47, 52, 54, 56, 58, 63, 65, 67, 74, 78, 80, 85, 87, 89, 96, 98$).

L'algoritmo deve avere complessità $O(n)$. **Motivare bene la correttezza e la complessità dell'algoritmo proposto**.

Utilizziamo una tabella bidimensionale di dimensione $(n+1) \times 10$ dove
$T[i,j]$ = il numero di combinazioni lunghe i che terminano con la cifra j.
La soluzione sarà in $\sum_{j=0}^{9}(T[i,j])$.

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

$$T[i,j] = \begin{cases} 0 & \text{se } i=0 \\ 1 & \text{se } i=1 \\ T[i-1][2]+T[i-1][4] & \text{se } j=1 \\ T[i-1][2]+T[i-1][6] & \text{se } j=3 \\ T[i-1][4]+T[i-1][8] & \text{se } j=7 \\ T[i-1][6]+T[i-1][8] & \text{se } j=9 \\ T[i-1][1]+T[i-1][3]+T[i-1][5] & \text{se } j=2 \\ T[i-1][3]+T[i-1][5]+T[i-1][9] & \text{se } j=6 \\ T[i-1][1]+T[i-1][5]+T[i-1][7] & \text{se } j=4 \\ T[i-1][2]+T[i-1][6]+T[i-1][8]+T[i-1][4] & \text{se } j=5 \\ T[i-1][5]+T[i-1][9]+T[i-1][0]+T[i-1][7] & \text{se } j=8 \\ T[i-1][8] & \text{se } i=0 \end{cases}$$

```python
def numeri(n):
    T=[[ 0 for _ in range(10)] for _ in range(n+1)]
    for i in range(n+1):
        for j in range(10):
            if i==0: T[i][j]=0
            elif i==1: T[i][j]=1
            elif j==0: T[i][j]=T[i-1][8]
            elif j==1: T[i][j]=T[i-1][2]+T[i-1][4]
            elif j==2: T[i][j]=T[i-1][1]+T[i-1][3]+T[i-1][5]
            elif j==3: T[i][j]=T[i-1][2]+T[i-1][6]
            elif j==4: T[i][j]=T[i-1][1]+T[i-1][5]+T[i-1][7]
            elif j==5: T[i][j]=T[i-1][2]+T[i-1][6]+T[i-1][8] +T[i-1][4]
            elif j==6: T[i][j]=T[i-1][3]+T[i-1][5]+T[i-1][9]
            elif j==7: T[i][j]=T[i-1][4]+T[i-1][8]
            elif j==8: T[i][j]=T[i-1][5]+T[i-1][9]+T[i-1][0] +T[i-1][7]
            elif j==9: T[i][j]=T[i-1][6]+T[i-1][8]
    return sum(T[n])

>>> numeri(1)
10
>>> numeri(2)
26
>>> numeri(3)
76
>>> numeri(4)
216
```

Complessità $\Theta(n)$

ESERCIZIO 2

Abbiamo una matrice M di interi di dimensione $n \times n$ con $n > 1$. Una *discesa* su questa matrice è una sequenza di n celle della matrice con i seguenti vincoli

- le celle appartengono a righe diverse della matrice
- la prima cella appartiene alla prima riga della matrice
- ogni altra cella è adiacente (in verticale o in diagonale) alla cella che la precede.

Ad esempio, per $M = \begin{pmatrix} 12 & 10 & 3 & 14 & 9 \\ 0 & 1 & 13 & 15 & 13 \\ 8 & 10 & 1 & 2 & 7 \\ 7 & 11 & 10 & 5 & 7 \\ 18 & 4 & 6 & 10 & 0 \end{pmatrix}$

sono evidenziate due possibili discese ($12, 1, 1, 11, 4$ e $3, 15, 2, 5, 6$).

Progettare un algoritmo che, data la matrice M, stampa tutte le possibili discese di M.

Ad esempio per $M = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ l'algoritmo stamperà le 4 discese: $1, 3 \quad 1, 4 \quad 2, 3 \quad 2, 4$.

L'algoritmo deve avere complessità $O(n^2 S(n))$ dove $S(n)$ è il numero di discese da stampare.

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

Algoritmo:

- per ognuno delle n celle della prima riga della matrice M eseguiremo una funzione di backtracking che enumera tutte le discese che partono da quel nodo.
- la funzione di backtraking poi produce le possibili discese di M facendo attenzione ad accodare a ciascuna discesa parziale solo le celle che possono allungare la discesa.

```
def es(M):
    for j in range(len(M)):
        esR(0,j,M,[])

def esR(i,j,M,sol):
    if i==len(M)-1:
        sol.append(M[i][j])
        print(sol)
        return
    sol.append(M[i][j])
    if j>0:
        esR(i+1,j-1,M,sol)
        sol.pop()
    esR(i+1,j,M,sol)
    sol.pop()
    if j<len(M)-1:
        esR(i+1,j+1,M,sol)
        sol.pop()

>>> M=[[1,2],[3,4]]
>>> es(M1)
[1, 3]
[1, 4]
[2, 3]
[2, 4]
```

- Nota che nell'albero di ricorsione prodotto dall'esecuzione di esR un nodo viene generato solo se porta ad una foglia da stampare.
- Possiamo quindi dire che la complessità dell'esecuzione di $esR(0, j, M, [])$ 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 $esR(0, j, M, [])$ è $O(nS(n))$.
- Otteniamo quindi che il costo di $es(n)$ è $O(n^2 S(n))$.