ESERCIZIO 1.

Considerate una griglia $n \times n$. Un cammino valido in questa griglia deve partire dalla cella in alto a sinistra di coordinate $(0,0)$ ed arrivare alla posizione in basso a destra di coordinate $(n-1,n-1)$.
E' possibile muoversi su questa griglia solo su celle adiacenti, andando di un passo verso destra oppure di un passo verso il basso. Oltretutto nel corso del cammino è vietato toccare le celle di coordinate (i,j) con $i > j$. Dato il valore di n vogliamo calcolare il numero di cammini validi.

Ad esempio:
per $n = 4$ la risposta deve essere 5, abbiamo infatti i seguenti possibili cammini:
$dddbbb \quad ddbdbb \quad ddbbdb \quad dbdbdb \quad dbddbb$
(dove d indica un passo verso destra e b un passo verso il basso).

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 $n \times n$.

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

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

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

La ricorrenza è corretta per i seguenti motivi:

- per $i > j$ si ha $T[i][j] = 0$ perché la cella (i, j) è irraggiungibile
- per i=0 si ha 1 perché posso raggiungere la cella $(0,j)$ a partire da $(0,0)$ solo spostandomi verso destra ad ogni passo
- In tutti gli altri casi si ha $T[i][j] = T[i-1][j] + T[i][j-1]$ perché posso raggiungere (i, j) sia dai cammini che provengono da $(i-1,j)$ che da quelli che provengono da $(i, j-1)$

Implementazione:

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



>>> es1(4)
5
>>> es1(5)
14
>>> es1(6)
42
```

Complessità $\Theta(n^2)$

ESERCIZIO 2.

Dati n e k, con $n \geq k$,definiamo *valida* una stringa di lunghezza n che ha come caratteri gli interi da 0 a $k-1$ e in cui ciascuno dei k caratteri compare almeno una volta. Ad esempio per $n = 4$ e $k = 3$ le stringhe valide sono le seguenti:
0012 0021 0102 0112 0120 0121 0122 0201 0210 0211 0212 0221 1002 1012 1020 1021 1022 1102 1120 1200 1201 1202 1210 1220 2001 2010 2011 2012 2021 2100 2101 2102 2110 2120 2201 2210

Trovate un algoritmo che dati n e k con $n \geq k$ stampi tutte le stringhe valide. Il vostro algoritmo deve avere complessità $O(k \cdot n \cdot S(n,k))$ dove $S(n,k)$ è il numero di stringhe valide 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 sol di lunghezza n sull'alfabeto $\{0, \ldots k-1\}$ con l'aggiunta di una funzione di taglio.
- Utiliziamo: una variabile t che riporta il numero di simboli che ancora non risultano inseriti nella stringa che si sta costruendo e un vettore di k componenti col dove $col[j]$ contiene il numero di simboli j presenti nella stringa fin'ora costruita.
- al passo i possiamo inserire qualunque simbolo se restano poi abbastanza posti per inserire i simboli non ancora presenti nella stringa (vale a dire se $n-i > t$). Se questo non vale allora dobbiamo necessariamente inserire un simbolo che ancora non compare nella stringa. Quindi:
  - nella posizione $sol[i]$ inserisco un qualunque simbolo j se e solo se $n-i > t$. Incremento $col[j]$ e decremento t se risulta $col[i] == 1$.
  - nella posizione $sol[i]$ inserisco solo i simboli per cui $col[j] == 0$ se e solo se $n-i <= t$. Incremento $col[j]$ e decremento t.

Implementazione

```python
def ex3(n,k):
    sol=[0]*n
    col=[0]*k
    ex3R(0,sol,col,k)

def ex3R(i,sol,col,t):
    if i==len(sol):
        print("".join(sol))
        return
    if len(sol)-i<=t:
        for j in range(len(col)):
            if col[j]==0:
                sol[i]=str(j)
                col[j]+=1
                ex3R(i+1,sol,col,t-1)
                col[j]-=1
    else:
         for j in range(len(col)):
                sol[i]=str(j)
                col[j]+=1
                if col[j]==1:
                    ex3R(i+1,sol,col,t-1)
                else:
                    ex3R(i+1,sol,col,t)
                col[j]-=1
```

```
>>> ex3(4,3)
0012
0021
0102
0112
0120
0121
0122
0201
0210
0211
0212
0221
1002
1012
1020
1021
1022
1102
1120
1200
1201
1202
1210
1220
2001
2010
2011
2012
2021
2100
2101
2102
2110
2120
2201
2210
```

- Nota che nell'albero di ricorsione prodotto dall'esecuzione di $es3$ un nodo viene generato solo se porta ad una foglia da stampare.
- Possiamo quindi dire che la complessità dell'esecuzione di $es3(n,k)$ richiederà tempo

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

  dove:

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

- Quindi il costo di $es3(n,k)$ è

$$\Theta(k\cdot n\cdot S(n,k))+\Theta(n\cdot S(n,k))=\Theta(n\cdot k\cdot S(n,k))$$