ESERCIZIO 1.

Dati tre interi positivi n, m e k, con $1 \leq , m, k \leq n$, vogliamo calcolare il numero di stringhe lunghe n sull'alfabeto $\{0,1,2\}$ dove il numero di occorrenze di 1 non supera m ed il numero di occorrenze di 2 non supera k.

Ad esempio per $n = 3$, $m = 1$ e $k = 2$ il numero è 19. Infatti le sole stringhe ternarie lunghe 3 con al più un 1 ed al più due 2 sono:
000, 001, 002, 010, 012, 020, 021, 022, 100, 102, 120, 122, 200, 201, 202, 210, 212, 220, 221

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

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

Utilizziamo una tabella di dimensioni $(n+1) \times (m+1) \times (k+1)$.

$T[i][j][t] =$ **numero di stringhe ternarie lunghe i con al più j simboli 1 e t simboli 2**

La soluzione al problema sarà nella cella $T[n][m][k]$.

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

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

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

La ricorrenza è corretta per i seguenti motivi:

- se $i = 0$ ho solo la stringa vuota e questa va bene qualunque sia j e t.
- se $i > 0$ e $j = t = 0$ allora l'unica stringa possibile è composta dai soli simboli 0
- se $i, t > 0$ e $j = 0$ allora le stringhe possibili possono terminare coi soli simboli 0 e 2. Quelle che terminano con 0 sono $T[i-1][0][t]$ quelle che terminano con 2 sono $T[i-1][0][t-1]$.
- se $i, j > 0$ e $t = 0$ allora le stringhe possibili possono terminare coi soli simboli 0 e 1. Quelle che terminano con 0 sono $T[i-1][j][0]$ quelle che terminano con 1 sono $T[i-1][j-1][0]$.
- se $i, j, t > 0$ allora le stringhe possibili possono terminare cocon uno qualunque dei 3 simboli. Quelle che terminano con 0 sono $T[i-1][j][t]$ quelle che terminano con 1 sono $T[i-1][j-1][t]$ e quelle che terminano con 2 sono $T[i-1][j][t-1]$.
-

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

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

Complessità $\Theta(nmk)$

```
>>> es1(3,0,3)
8
>>> es1(3,1,0)
4
>>> es1(3,2,1)
19
>>> es1(3,3,3)
27
>>> es1(3,2,2)
25
```

ESERCIZIO 2

Progettare un algoritmo che, dato un intero n, stampa tutte le stringhe di lunghezza $2 \cdot n$ sull'alfabeto $\{1,2,3\}$ tali che il numero di cifre dispari presenti nella prima metà della stringa è lo stesso del numero di cifre dispari presenti nella seconda metà.

Ad esempio per $n = 2$ l'algoritmo deve stampare, non necessariamente nello stesso ordine, le seguenti 33 stringhe:

1111 1113 1131 1133 1221 1223 1212 1232 1311 1313 1331
1333 2121 2123 2112 2132 2222 2321 2323 2312 2332 3111
3113 3131 3133 3221 3223 3212 3232 3311 3313 3331 3333

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:

- nota che le stringhe che vogliamo stampare sono quelle che hanno lo stesso numero di simboli 2 nella prima e nella seconda metà
- Utilizziamo un algoritmo di backtracking per la stampa di tutte le stringhe ternarie lunghe $2n$ con l'aggiunta di funzioni di taglio.
- per la generazione dei primi n bit della stringa tutti i nodi vengono generati e viene anche utilizzato un contatore *tot* che riporta il numero di 2 inseriti fino a quel momento.
- dopo l'inserimento dei primi n bit il contatore *tot* viene decrementato ad ogni inserimento di un simbolo 2 e la generazione dei nodi è soggetta a due funzioni di taglio:
  - un 1 o un 3 viene inserito nella stringa solo se restano poi sufficienti bit da settare per portare a zero il valore di *tot*
  - un 2 viene inserito nella stringa solo se il contatore *tot* è positivo.
- grazie alle due funzioni di taglio la soluzione parziale via via costruita è sempre un prefisso di una soluzione da stampare.

Implementazione:

```
def BT(n,i=0,tot=0,sol=[]):
    if i==2*n:
        print (''.join(sol))
    else:
        if i<n:
            for x in ['1','2','3']:
                sol.append(x)
                if x =='2':
                    BT(n,i+1,tot+1,sol)
                else:
                    BT(n,i+1,tot,sol)
                sol.pop()
        else:
            if tot>0:
                sol.append('2')
                BT(n,i+1,tot-1,sol)
                sol.pop()
            if 2*n-(i+1)>=tot:
                for x in ['1','3']:
                    sol.append(x)
                    BT(n,i+1,tot,sol)
                    sol.pop()
```

- 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 $bk(n, 0, 0, sol)$ richiederà tempo

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

dove:

  - $h = 2 \cdot n$ 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 $BT(n, 0, 0, [\,])$ è $O(nS(n))$.

```
>>> BT(2)
1111
1113
1131
1133
1221
1223
1212
1232
1311
1313
1331
1333
2121
2123
2112
2132
2222
2321
2323
2312
2332
3111
3113
3131
3133
3221
3223
3212
3232
3311
3313
3331
3333
```

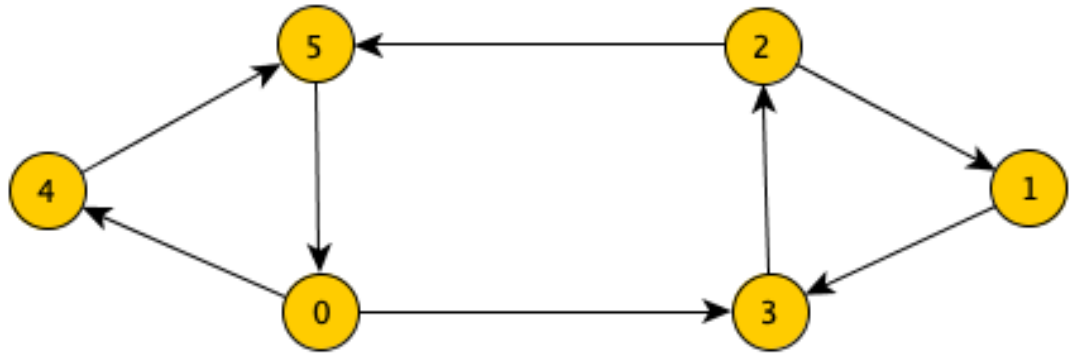
ESERCIZIO 3.

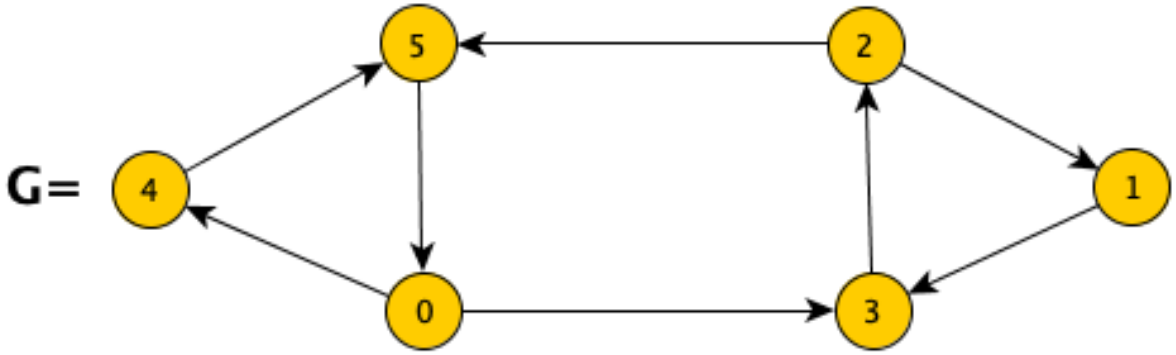
Considerate il grafo G in figura.

1. Quale è il numero minimo di archi da eliminare da G perché il grafo abbia sort topologici? **Motivare BENE la vostra risposta.**

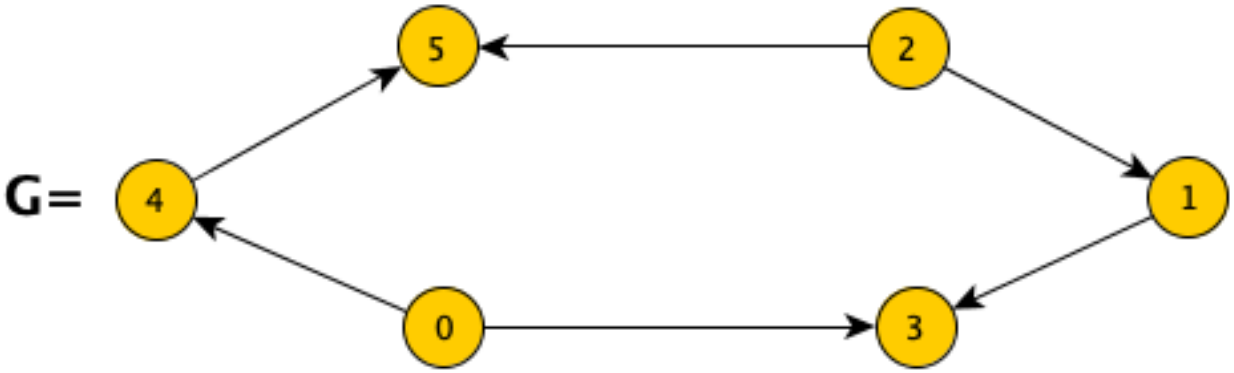
3. Eliminate da grafo G il numero minimo di archi perché questi risulti avere sort topologici e applicate al grafo così ottenuto l'algoritmo delle sorgenti visto a lezione . Nell'eseguire l'algoritmo qualora risultino più sorgenti tra cui scegliere prendete sempre quella di indice minimo. Qual' è il sort topologico che si ottiene in questo modo?

G=





nel grafo sono presenti almeno due cicli disgiunti: il ciclo (0,4,5) ed il ciclo (2,3,2).
Devo quindi eliminare dal grafo almeno due archi per rompere questi cicli.
Se elimino gli archi (5,0) e (3,2) ottengo il grafo aciclico:



La risposta alla prima domanda è quindi 2.

Applicando ora l'algoritmo delle sorgenti per la ricerca del sort topologico e scegliendo di volta in volta la sorgente di indice minimo si ottiene il sort topologico**:**

0, 2, 1, 3, 4, 5