ESERCIZIO 1.

Dato un intero n, con $n \geq 2$, vogliamo contare il numero di modi in cui è possibile ottenere n partendo dal numero 2 e potendo effettuare le sole 3 operazioni di incremento di 1, prodotto per 2 e prodotto per 3.

Ad esempio per $n = 10$ la risposta è 9. Infatti le sequenze possibili sono:
$(2,3,4,5,6,7,8,9,10)$, $(2,3,4,5,10)$, $(2,3,4,8,9,10)$, $(2,3,6,7,8,9,10)$,
$(2,3,9,10)$, $(2,4,5,10)$, $(2,4,5,6,7,8,9,10)$, $(2,4,8,9,10)$, $(2,6,7,8,9,10)$.

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

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

Utilizziamo una tabella di dimensioni n.

$T[i] =$ **numero di modi in cui è possibile ottenere** i **partendo da** 2

La soluzione al problema sarà $T[n]$.

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

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

$$T[i] = \begin{cases} 1 & \text{se } i = 2 \text{ o } i = 3 \\ T[i-1] + T[i/2] + T[i/3] & \text{se } i \text{ è divisibile per 6} \\ T[i-1] + T[i/2] & \text{se } i \text{ è divisibile per 2} \\ T[i-1] + T[i/3] & \text{se } i \text{ è divisibile per 3} \\ T[i-1] & \text{altrimenti} \end{cases}$$

La ricorrenza è corretta per i seguenti motivi:

- se $i = 2$ posso ottenere il numero solo non eseguendo operazioni
- Se $i = 3$ posso ottenere il numero in un solo modo: incrementando 2 di 1
- se il numero i è divisibile per 6 posso ottenerlo a partire $i - 1$, ma anche da $i/2$ ed anche da $i/3$ eseguendo l'opportuna operazione di incremento.
- se il numero i è divisibile per 2 e non per 3 posso ottenerlo solo a partire $i - 1$ o da $i/2$ eseguendo l'opportuna operazione di incremento.
- se il numero i è divisibile per 3 e non per 2 posso ottenerlo solo a partire $i - 1$ o da $i/3$ eseguendo l'opportuna operazione di incremento.
- se il numero i non è divisibile per 2 e non è divisibile per 3 posso ottenerlo solo a partire $i - 1$ eseguendo l'operazione di incremento di 1.

$$T[i] = \begin{cases} 1 & \text{se } i = 2 \text{ o } i = 3 \\ T[i-1] + T[i/2] + T[i/3] & \text{se } i \text{ è divisibile per 6} \\ T[i-1] + T[i/2] & \text{se } i \text{ è divisibile per 2} \\ T[i-1] + T[i/3] & \text{se } i \text{ è divisibile per 3} \\ T[i-1] & \text{altrimenti} \end{cases}$$

```python
def es1(n):
    T=[0]*(n+1)
    T[2]=1
    T[3]=1
    for i in range(4,n+1):
        T[i]=T[i-1]
        if i%2==0: T[i]+=T[i//2]
        if i%3==0: T[i]+=T[i//3]
    return T[n]
```

Complessità $\Theta(n)$

```
>>> es1(10)
9
>>> es1(6)
4
>>> es1(20)
47
```

ESERCIZIO 2

Progettare un algoritmo che dato un intero n stampa tutte le matrici binarie quadrate $n \times n$ dove il numero di zeri presenti in ciascuna colonna è minore o uguale al numero di uni della colonna.

Ad esempio per $n = 2$ l'algoritmo deve stampare, non necessariamente nello stesso ordine, le seguenti 9 matrici:

$$\begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

L'algoritmo proposto deve avere complessità $O(n^2 S(n))$ 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 matrici binarie sol di dimensione $n \times n$ con l'aggiunta di una funzione di taglio.
- In posizione $sol[i][j]$ non può esserci il simbolo 0 se nella colonna della matrice costruita fino a quel momento sono stati inseriti $n//2$ zeri mentre il simbolo 1 può sempre essere inserito.
- grazie alla funzione di taglio la soluzione parziale via via costruita è sempre un prefisso di una soluzione da stampare.
- Per calcolare efficientemente la funzione di taglio useremo un arry C di n contatori dove $C[i]$ contiene il numero di zeri inseriti nella colonna j della matrice fin'ora costruita.

Implementazione:

```
def es2(n):
    sol=[[0 for j in range(n)] for i in range(n)]
    C=[0 for i in range(n)]
    BT(0,0,n,C,sol)

def BT(i,j,n,C,sol):
    if i==n:
        for riga in range(n):
            print (sol[riga])
        print()
    else:
        if C[j]<n//2:
            sol[i][j]=0
            C[j]+=1
            if j<n-1:
                BT(i,j+1,n,C,sol)
            else: BT(i+1,0,n,C,sol)
            C[j]-=1
        sol[i][j]=1
        if j<n-1:
            BT(i,j+1,n,C,sol)
        else: BT(i+1,0,n,C,sol)
```

```
>>> es2(2)
[0, 0]
[1, 1]

[0, 1]
[1, 0]

[0, 1]
[1, 1]

[1, 0]
[0, 1]

[1, 0]
[1, 1]

[1, 1]
[0, 0]

[1, 1]
[0, 1]

[1, 1]
[1, 0]

[1, 1]
[1, 1]
```

- 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 $BT(0, 0, n, C, sol)$ richiederà tempo

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

  dove:

  - $h = n^2$ è l'altezza dell'albero.
  - $f(n) = O(1)$ è il lavoro di un nodo interno.
  - $g(n) = O(n^2)$ è il lavoro di una foglia

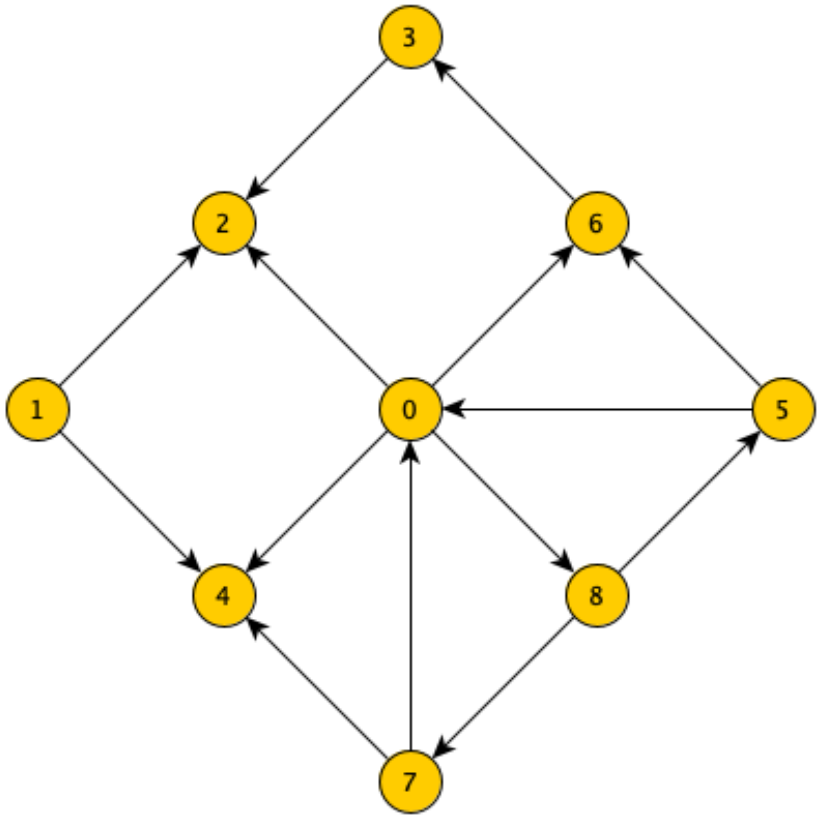
- Quindi il costo di è $O(n^2 S(n))$.

ESERCIZIO 3

Considerate il grafo G in figura.

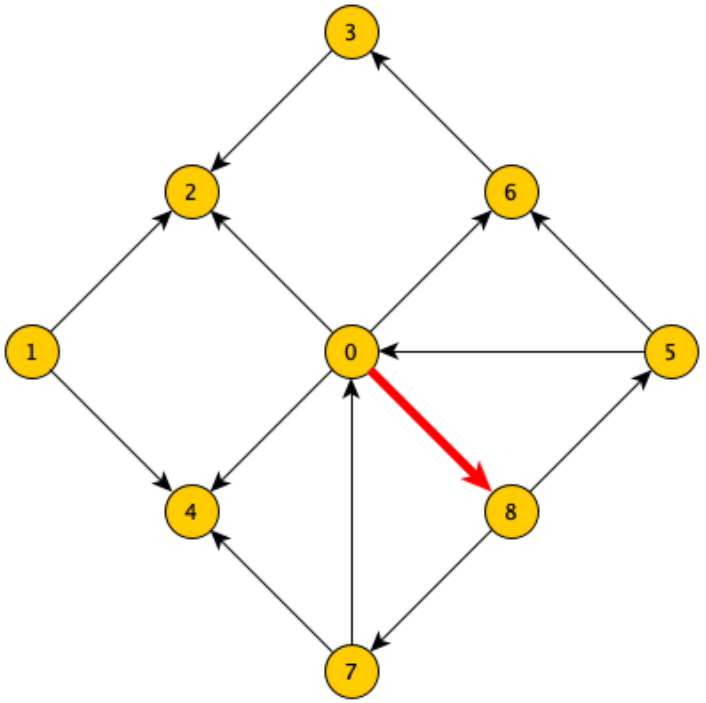
1. Quale è il numero minimo di archi da eliminare da G perché il grafo abbia sort topologici? **Motivare BENE la vostra risposta.**
2. Eliminate da grafo G il numero minimo di archi perché questi risulti avere sort topologici e applicate al grafo così ottenuto l'algoritmo per la ricerca del sort topologico basato sulla visita in profondità. Nell'eseguire l'algoritmo qualora risultino più nodi tra cui scegliere per proseguire la visita prendete sempre quella di indice minimo. Qual' è il sort topologico che si ottiene in questo modo?

G=

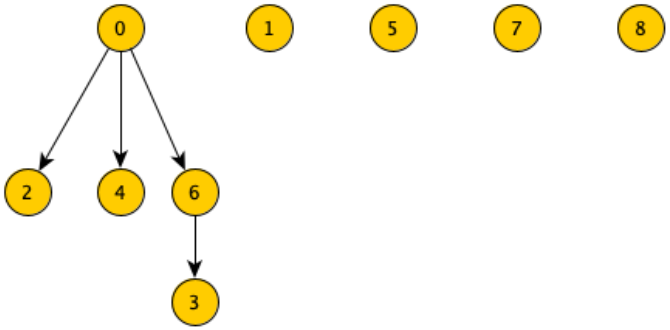


Un grafo ha sort topologici se e solo se è aciclico. Nel grafo sono presenti due cicli: (0,8,5) e (0,8,7) Per avere sort topologici devo quindi eliminarli dal grafo, i cicli non sono disgiunti ma hanno in comune l'arco (0,8) basterà quindi eliminare quell'arco per rendere il grafo aciclico

G=



La risposta alla prima domanda è quindi 1.



Applicando ora l'algoritmo della visita in profondità per la ricerca del sort topologico la visita dei nodi termina secondo quest'ordine:

2-4-3-6-0-1-5-7-8

La risposta alla seconda domanda è quindi : 8-7-5-1-0-6-3-4-2