ESERCIZIO 1.

Data una sequenza S di n cifre decimali, vogliamo calcolare la lunghezza massima per una sottosequenza non decrescente che contiene i tre simboli 0, 1 e 2.

Ad esempio:

- per $S = 5,0,6,7,2,2,1,0,1$ la risposte deve essere 0 (non esistono infatti in S sottosequenze nondecrescenti che contengono i tre simboli 0, 1, e 2).
- per $S = 0,1,2,5,6,1,4,1,2,2,2,1,5,2$ la risposta deve essere 8 (infatti la sottosequenza non decrescente più lunga con tutti e tre i simboli è $0,1,\_,\_,\_,1,\_,1,2,2,2,\_,\_,2$.

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 $n \times 3$.

$T[i][j] =$ **la lunghezza massima per una sottosequenza non decrescente che termina con il simbolo** j **in** $S_0, S_1, \ldots S_i$

La soluzione al problema sarà $T[n-1][2]$.

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

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

$$T[i][j] = \begin{cases} 1 & \text{se } i = j = 0 \text{ AND } S_0 = 0 \\ 0 & \text{se } i = 0 \\ T[i-1][j] & \text{se } j \neq S_i \\ T[i-1][0] + 1 & \text{se } j = 0 \\ 0 & \text{se } j = 1 \text{ AND } T[i-1][0] = 0 \text{ AND } T[i-1][1] = 0 \\ \max\{\ T[i-1][0],\ T[i-1][1]\ \} + 1 & \text{se } j = 1 \\ 0 & \text{se } j = 2 \text{ AND } T[i-1][1] = 0 \text{ AND } T[i-1][2] = 0 \\ \max\{\ T[i-1][1],\ T[i-1][2]\ \} + 1 & \text{se } j = 2 \end{cases}$$

La ricorrenza è corretta per i seguenti motivi:

- per $i = 0$ si ha $T[0][j] \neq 0$ solo se $j = S_0 = 0$

consideriamo ora i casi $i > 0$:

- se $S_i \neq j$ allora il simbolo S_i non contribuisce in alcun modo alla sottosequenza di lunghezza massima in $S_0, S_1, \ldots S_i$ che termina con j quindi deve aversi $T[i][j] = T[i-1][j]$.
- se $S_i = 0$ allora il simbolo S_i contribuisce alla sottosequenza di lunghezza massima che termina con 0 .
- se $S_i = 1$ allora la sottosequenza più lunga che termina con 1 è diversa da zero solo se il simbolo S_1 può agganciarsi ad una sottosequenza che termina con 0 o ad una sottosequenza che termina con 1, deve quindi aversi $T[i-1][0] \neq 0$ o $T[i-1][1] \neq 0$ e si aggancerà alla più lunga delle due.
- se $S_i = 2$ allora la sottosequenza più lunga che termina con 2 è diversa da zero solo se il simbolo S_1 può agganciarsi ad una sottosequenza che termina con 1 o ad una sottosequenza che termina con 2, deve quindi aversi $T[i-1][1] \neq 0$ o $T[i-1][2] \neq 1$ e si aggancerà alla più lunga delle due.

Implementazione:

```
def es1(S):
    n=len(S)
    T=[[0,0,0] for _ in range(n)]
    if S[0]==0: T[0][0]=1
    for i in range(1,n):
        for j in [0,1,2]:
            if S[i]!=j:
                T[i][j]=T[i-1][j]
            elif j==0:
                T[i][0]=T[i-1][0]+1
            elif j==1:
                T[i][1] = max(T[i-1][0], T[i-1][1])
                if T[i][1]: T[i][1]+=1
            elif j==2:
                T[i][2] = max(T[i-1][1], T[i-1][2])
                if T[i][2]: T[i][2]+=1
    return T[n-1][2]
```

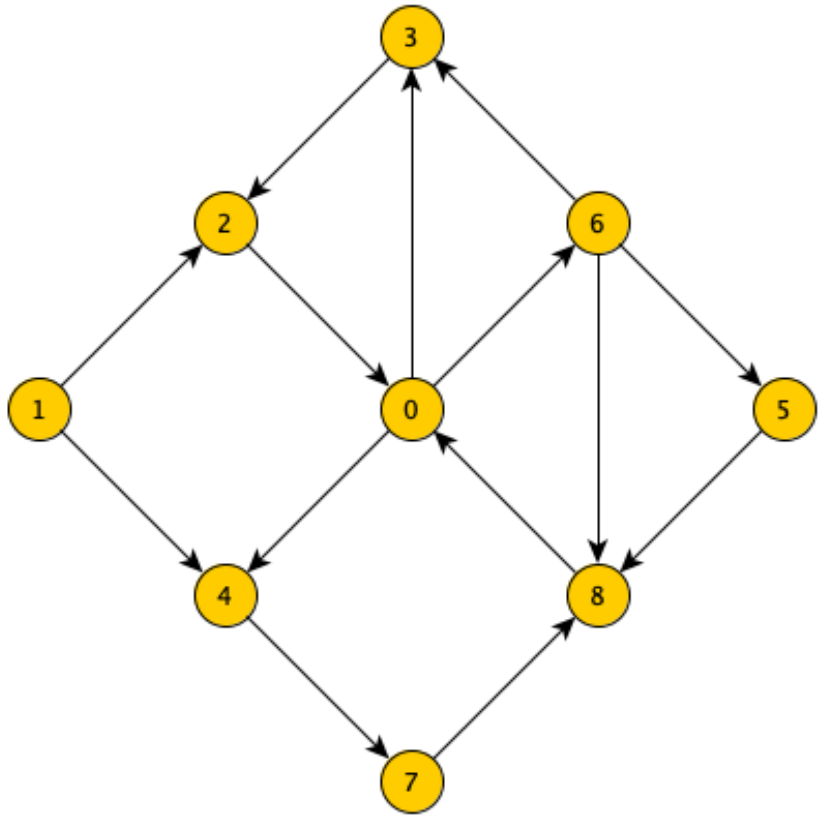
Complessità $\Theta(n)$

ESERCIZIO 2

Considerate il grafo G in figura.

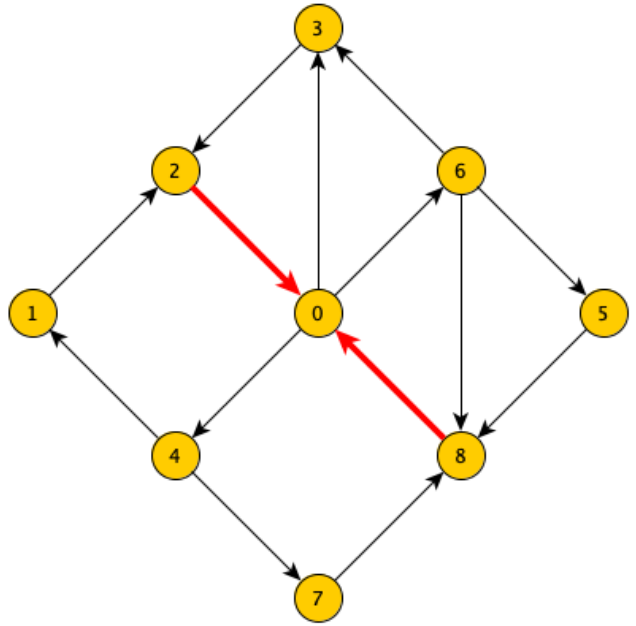
1. Quale è il numero minimo di archi da eliminare da G perché il grafo abbia sort topologici? **Motivare BENE la vostra risposta.**
2. Eliminate da grafo G il numero minimo di archi perché questi risulti avere sort topologici e applicate al grafo così ottenuto l'algoritmo per la ricerca del sort topologico 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=

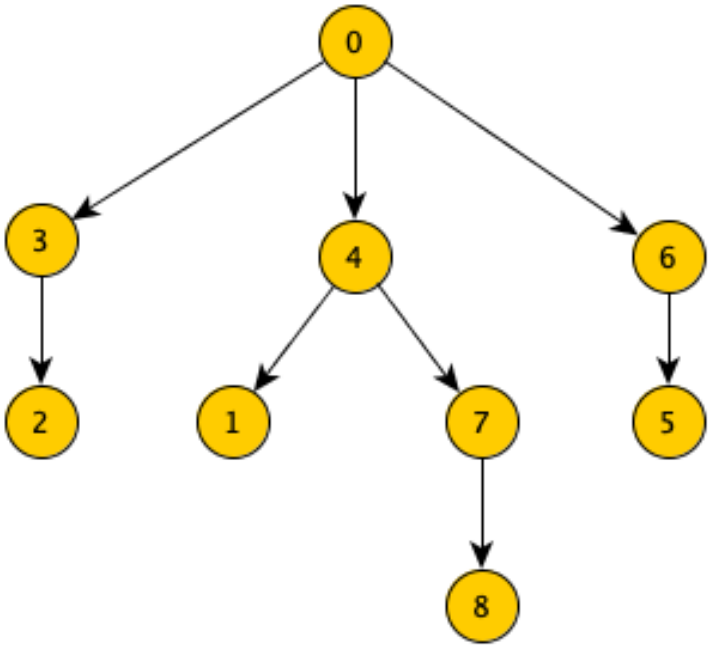


Un grafo ha sort topologici se e solo se è aciclico. Nel grafo sono presenti due cicli arco-disgiunti:(0,3,2) e (0,6,8). Per avere sort topologici devo quindi eliminare almeno due archi dal grafo (uno per ciclo). Se elimino gli archi (2,0) e (8,0) il grafo risulta aciclico.

G=



La risposta alla prima domanda è quindi 2.



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

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

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

ESERCIZIO 3

Progettare un algoritmo che, dato un intero n, stampa tutte le matrici binarie $n \times n$ con la proprietà che nella riga i, $0 \leq i < n$, della matrice sono presenti esattamente i uni. L'algoritmo proposto deve avere complessità $O(n^2 S(n))$ dove $S(n)$ è il numero di matrici da stampare.

Ad esempio per $n = 3$ l'algoritmo deve stampare, non necessariamente nello stesso ordine, le seguenti 9 matrici:

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

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

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 funzioni di taglio.
- Utilizzo un array R di n componenti dove $R[i]$ contiene il numero di uni inseriti fino a quel momento nella riga i della matrice che si sta costruendo.
- al passo i:
  - nella cella $sol[i][j]$ inserisco 1 se e solo se $R[i] < i$ e richiamo poi la funzione sulla cella successiva da riempire dopo aver incrementato $R[i]$
  - nella cella $sol[i][j]$ inserisco 0 se e solo se $j - R[i] < n - i$ e richiamo poi la funzione sulla cella successiva da riempire

Implementazione

```
def es3(n):
    sol=[[0]*n  for _ in range(n)]
    R=[0]*n
    es3b(n,0,0,sol,R)


def es3b(n,i,j,sol,R):
    if i==n:
        for r in range(n):
            print(sol[r])
        print()
        return
    if j-R[i] < n-i:
        sol[i][j]=0
        if j<n-1:
            es3b(n,i,j+1,sol,R)
        else:
            es3b(n,i+1,0,sol,R)
    if R[i]<i :
        sol[i][j]=1
        R[i]+=1
        if j<n-1:
            es3b(n,i,j+1,sol,R)
        else: es3b(n,i+1,0,sol,R)
        R[i]-=1
```

- 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 $es3(n)$ 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.
  - $g(n) = O(n^2)$ è il lavoro di una foglia

- Quindi il costo di $es3(n)$ è $O(n^2 S(n))$.