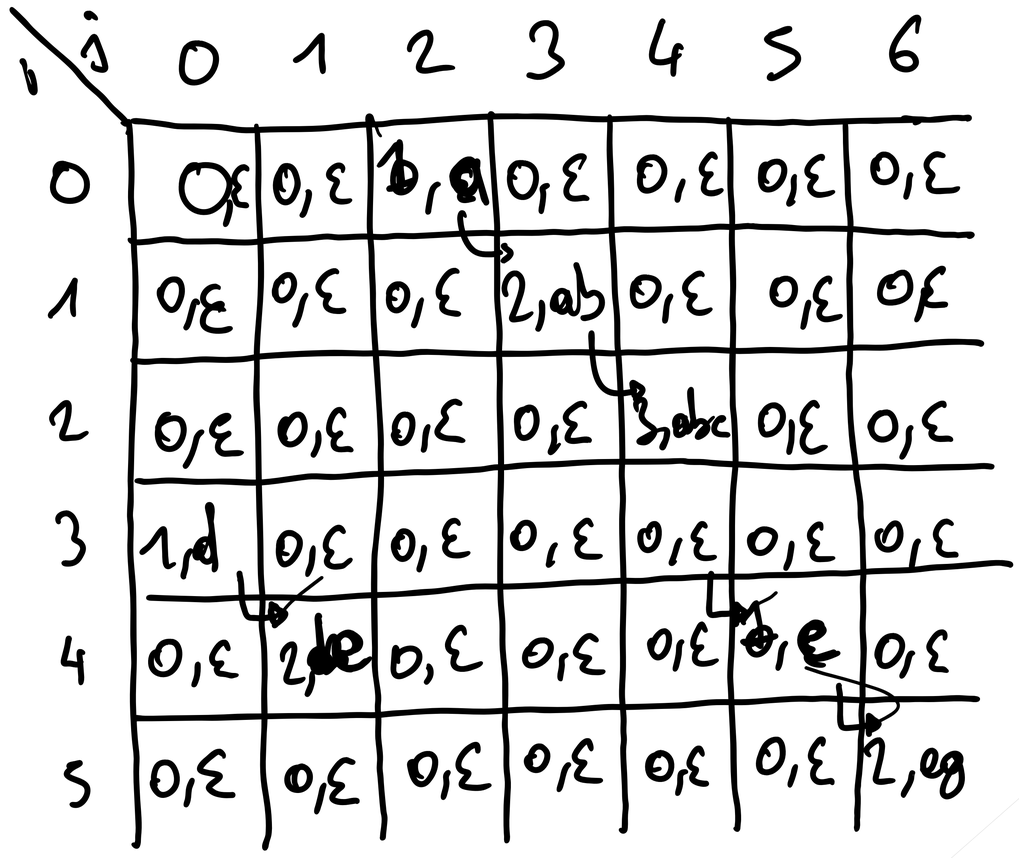
```
def estSousChaine(s,t):
    # retourne i tq s commence à t[i]
    # -1 sinon
    for i in range(len(t)-len(s)+1):
        if s==t[i:i+len(s)]: return i
    return -1
```

Complexité $O(|s|.|t|)$

Algo pour trouver des plus longues sous-chaînes communes de s et t
On teste toutes les sous chaînes de s, si elles sont sous chaines de t
et on prend une des plus longues

Complexité : $O(\underbrace{|s|^2}_{\text{nbr ss chaines}} . \underbrace{|s|.|t|}_{\text{coup de chaque test}}) = O(|s|^3.|t|)$



s = a b c d e g (0 1 2 3 4 5)
t = d c a b c e g (0 1 2 3 4 5 6)

lplsce(i,j)
plscc(i,j)

$$lplsce(i,j) = \begin{cases} 0 & \text{si } s[i] \neq t[j] \\ 1 & \text{si } s[i]=t[j] \text{ et } (i=0 \text{ ou } j=0) \\ 1+lplsce(i-1,j-1) & \text{sinon} \end{cases}$$

plscc(i,j) = plscc(i-1,j-1) . s[i]

Complexité $O(|s|.|t|)$

TD6

Ex1 :

si mot vide

		↓	a	n	a	n	a	s
i \ j		0	1	2	3	4	5	6
si mot vide →	0	0	1	2	3	4	5	6
b	1	1	1	2	3	4	5	6
a	2	2	1	2	2	3	4	5
n	3	3	2	1	2	2	3	4
a	4	4	3	2	1	2	2	3
n	5	5	4	3	2	1	2	3
e	6	6	5	4	3	2	2	3

10 = min +1

● 1 car on est dans le cas 2 : a == a

```
def distance(a,b):
    L = [[ j for j in range (len(b)+1)]]
        + [ [i] + [0 for k in range(1, len(a)+1)]   # ici on a initialisé les lignes et colonnes

    for i in range (1, len(a)+1):
        for j in range (1, len(b)+1):
            if a[i-1] == b[j-1] : L[i][j] = L[i-1][j-1]
            else : L[i][j] = 1+min(L[i-1][j-1], L[i-1][j-1], L[i][j-1])
    return L[len(a)][len(b)]
```

Complexité $O(|a| \cdot |b|)$