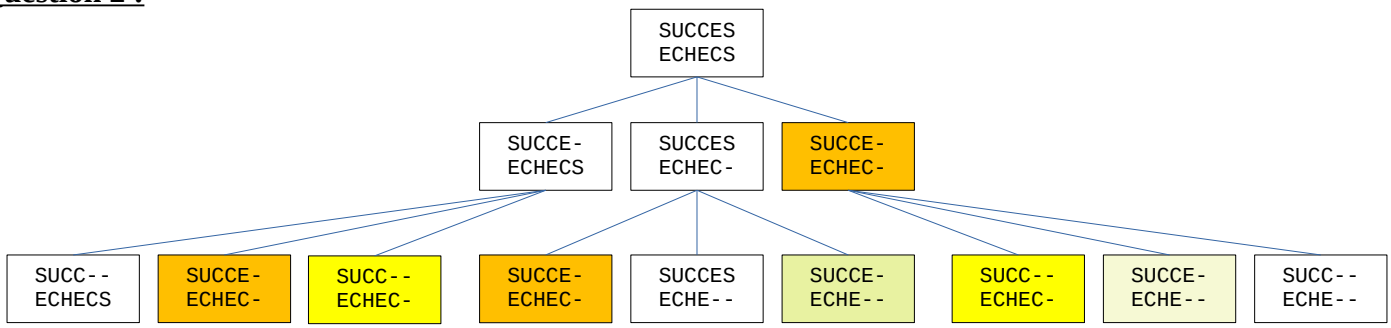
Question 1 :

```
S - U C C E - S
- E - C H E C S
```

- suppression de S : UCCES (coût = 1)
- insertion de E : EUCCES (coût = 1)
- suppression de U : ECCES (coût = 1)
- correspondance de C : ECCES
- substitution de C par H : ECHES (coût = 1)
- correspondance de E : ECHES
- insertion de C : ECHECS (coût = 1)
- correspondance de S : ECHECS

dE(SUCCES, ECHECS) = 5
Cet alignement est moins bon que le précédent.

Question 2 :



Question 3 :

```
d[i][j] = min(d[i-1][j], d[i][j-1], d[i-1][j-1] + sub)
```

Question 4 :

substitution : case au dessus à gauche en diagonale + 1
suppression : case au dessus + 1
insertion : case à gauche + 1
correspondance : case au dessus à gauche en diagonale + 0

Question 5 :

		0	1	2	3	4	5	6
		ε	E	C	H	E	C	S
0	ε	0	1	2	3	4	5	6
1	S	1	1	2	3	4	5	5
2	U	2	2	2	3	4	5	6
3	C	3	3	2	3	4	4	5
4	C	4	4	3	3	4	4	5
5	E	5	4	4	4	3	4	5
6	S	6	5	4	4	4	4	4

Question 6 :

		0	1	2	3	4	5	6
		ε	C	H	I	E	N	S
0	ε	0	1	2	3	4	5	6
1	N	1	1	2	3	4	4	5
2	I	2	2	2	2	3	4	5
3	C	3	2	3	3	3	4	5
4	H	4	3	2	3	4	4	5
5	E	5	4	3	3	3	4	5

Question 7 :

```
def dE(t1, t2):
    # ÉTAPE 1 : Création et initialisation du tableau
    # création du tableau D
    n1 = len(t1)
    n2 = len(t2)
    D = [[0] * (n2+1) for i in range(n1+1)]

    # initialisation première colonne et première ligne
    for i in range(n1 + 1):
        D[i][0] = i
    for j in range(n2 + 1):
        D[0][j] = j

    # ÉTAPE 2 : Remplissage du reste du tableau
    for i in range(1, n1+1):
        for j in range(1, n2+1):
            if t1[i-1] == t2[j-1]:  # correspondance
                cout_sub = 0
            else:  # substitution
                cout_sub = 1
            D[i][j] = min(D[i-1][j]+1, D[i][j-1] + 1, D[i-1][j-1] +
cout_sub)

    # ÉTAPE 3 : Le résultat est dans la case en bas à droite
    return D[-1][-1], D
```

Question 8 :

```
>>> dE('SUCCES', 'ECHECS')
(4, [[0, 1, 2, 3, 4, 5, 6], [1, 1, 2, 3, 4, 5, 5], [2, 2, 2, 3, 4, 5, 6],
[3, 3, 2, 3, 4, 4, 5], [4, 4, 3, 3, 4, 4, 5], [5, 4, 4, 4, 3, 4, 5], [6,
5, 5, 5, 4, 4, 4]])
>>> dE('NICHE', 'CHIENS')
(5, [[0, 1, 2, 3, 4, 5, 6], [1, 1, 2, 3, 4, 4, 5], [2, 2, 2, 2, 3, 4, 5],
[3, 2, 3, 3, 3, 4, 5], [4, 3, 2, 3, 4, 4, 5], [5, 4, 3, 3, 3, 4, 5]])
```

```
>>> dE('SOUSTRACTION', 'MULTIPLICATION')
(8, [[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14], [1, 1, 2, 3, 4,
5, 6, 7, 8, 9, 10, 11, 12, 13, 14], [2, 2, 2, 3, 4, 5, 6, 7 , 8, 9, 10,
11, 12, 12, 13], [3, 3, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 , 12, 13, 13], [4,
4, 3, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14], [5, 5, 4, 4, 3, 4, 5, 6,
7, 8, 9, 10, 11, 12, 13], [6, 6, 5, 5, 4 , 4, 5, 6, 7, 8, 9, 10, 11, 12,
13], [7, 7, 6, 6, 5, 5, 5, 6, 7, 8 , 8, 9, 10, 11, 12], [8, 8, 7, 7, 6, 6,
6, 6, 7, 7, 8, 9, 10, 11, 12], [9, 9, 8, 8, 7, 7, 7, 7, 7, 8, 8, 8, 9, 10,
11], [10, 10, 9, 9, 8, 7, 8, 8, 7, 8, 9, 9, 8, 9, 10], [11, 11, 10, 10, 9,
8, 8, 9, 8, 8, 9, 10, 9, 8, 9], [12, 12, 11, 11, 10, 9, 9, 9, 9, 9, 9, 10,
10, 9, 8]])
```

Question 9 :

En passant par la case (1, 1)

```
S U C C E - S
E - C H E C S
```

		0	1	2	3	4	5	6
		ε	E	C	H	E	C	S
0	ε	0	1	2	3	4	5	6
1	S	1	1	2	3	4	5	5
2	U	2	2	2	3	4	5	6
3	C	3	3	2	3	4	4	5
4	C	4	4	3	3	4	4	5
5	E	5	4	4	4	3	4	5
6	S	6	5	4	4	4	4	4

Question 10 :

Il y a trois alignements optimaux entre `NICHE` et `CHIENS`.

```
- N I C H E
C H I E N S
```

```
N - I C H E
C H I E N S
```

```
N I C H - E - -
- - C H I E N S
```

		0	1	2	3	4	5	6
		ε	C	H	I	E	N	S
0	ε	0	1	2	3	4	5	6
1	N	1	1	2	3	4	4	5
2	I	2	2	2	2	3	4	5
3	C	3	2	3	3	3	4	5
4	H	4	3	2	3	4	4	5
5	E	5	4	3	3	3	4	5

Question 11 :

```
def solutions(t1, t2):
    # ÉTAPE 1 : Création et initialisation du tableau
    # création du tableau D
    n1 = len(t1)
    n2 = len(t2)
    D = [ [0] * (n2+1) for i in range(n1+1) ]
    sol = [[ [] for _ in range(n2+1) ] for i in range(n1+1)]

    # initialisation première colonne et première ligne
    for i in range(n1 + 1):
        D[i][0] = i
    for j in range(n2 + 1):
        D[0][j] = j
    for i in range(1,n1 + 1):
        sol[i][0] = [(i-1,0)]
    for j in range(1,n2 + 1):
        sol[0][j] = [(0,j-1)]

    # ÉTAPE 2 : Remplissage du reste du tableau
    for i in range(1, n1+1):
        for j in range(1, n2+1):
            if t1[i-1] == t2[j-1]:  # correspondance
                cout_sub = 0
            else:  # substitution
                cout_sub = 1
            D[i][j] = min(D[i-1][j] + 1, D[i][j-1] + 1, D[i-1][j-1] + cout_sub)
            if D[i][j] == (D[i-1][j] + 1):
                sol[i][j].append((i-1, j))
            if D[i][j] == (D[i][j-1] + 1):
                sol[i][j].append((i, j-1))
            if D[i][j] == (D[i-1][j-1] + cout_sub):
                sol[i][j].append((i-1, j-1))

    # ÉTAPE 3 : Le résultat est dans la case en bas à droite
    return sol

def alignement_optimal(t1, t2):
    sol = solutions(t1, t2)

    def creation_des_alignements(sol, i, j):
        if (i, j) == (0, 0):
            return [[(0, 0)]]

        alignements = []
        for suivant in sol[i][j]:
            res = creation_des_alignements(sol, suivant[0], suivant[1])
            for chaine in res:
                alignements.append( chaine + [(i, j)])
        return alignements

    return creation_des_alignements(sol, len(t1), len(t2))

#alignement_optimal('SUCCES', 'ECHECS')
alignement_optimal('NICHE', 'CHIENS')
```

```
[[(0, 0), (1, 0), (2, 0), (3, 1), (4, 2), (4, 3), (5, 4),(5, 5),(5, 6)],
[(0, 0), (1, 1), (1, 2), (2, 3), (3, 4), (4, 5), (5, 6)],
[(0, 0), (0, 1), (1, 2), (2, 3), (3, 4), (4, 5), (5, 6)]]
```

Question 12 :

```
# Version récursive avec memoïsation
def dE(t1, t2, memo):
    n1, n2 = len(t1), len(t2)
    if n1 == 0:
        return n2
    elif n2 == 0:
        return n1
    else:
        if t1[-1] == t2[-1]:
            sub = 0
        else:
            sub = 1

        if memo[n1 -1][n2] is None:
           memo[n1 -1][n2] = dE(t1[0:-1], t2, memo)
        if memo[n1][n2 -1] is None:
           memo[n1][n2 -1] = dE(t1, t2[0:-1], memo)
        if memo[n1 -1][n2 -1] is None:
           memo[n1 -1][n2 -1] = dE(t1[0:-1], t2[0:-1], memo)

        return min(memo[n1 -1][n2] + 1,
                   memo[n1][n2 -1] + 1,
                   memo[n1-1][n2 -1] + sub)

def init_dE(t1, t2):
    n1, n2 = len(t1), len(t2)
    memo = [ [None] * (n2+1) for i in range(n1+1) ]
    for i in range(n1 + 1):
        memo[i][0] = i
    for j in range(n2 + 1):
        memo[0][j] = j
    memo[-1][-1] = dE(t1, t2, memo)
    return memo[-1][-1], memo

init_dE('SUCCES', 'ECHECS')
```