Proposition de correction

Exercice 1

Q1

Q2.1

```
def hauteur_pile(P):
    Q = creer_pile_vide()
    n = 0
    while not(est_vide(P)):
        n += 1
        x = depiler (P)
        empiler(Q, x)
    while not(est_vide(Q)):
        x = depiler (Q)
        empiler(P, x)
    return n
```

Q2.2

```
def max_pile(P : object, i : int) -> int:
    """ renvoie la position j de l'élément maximum parmi les i derniers éléments empilés de P.
    Le sommet de la pile est 1. P conserve son état d'origine. """
    j = 0
    if not est_vide(P):
        Q    = creer_pile_vide()
        max = depiler (P)
        empiler(Q, max)
        j, n = 1, 1
        i   -= 1
        while not est_vide(P) and i > 0:
            n += 1
            x   = depiler (P)
            empiler(Q, x)
            if x > max:
                max = x
```

```
            j  = n
        i -= 1
    while not est_vide(Q):
        empiler(P, depiler (Q))
    return j
```

Q3

```
def retourner(P : object, j : int):
    """ inverse l'ordre des j derniers éléments empilés de P """
    Q = creer_pile_vide()
    while not est_vide(P) and j > 0:
        empiler(Q, depiler(P))
        j -= 1
    tmp = creer_pile_vide()
    while not est_vide(Q):  # inverse l'empilement
        empiler(tmp, depiler(Q))
    while not est_vide(tmp):
        empiler(P, depiler (tmp))
```

Q4

```
def tri_crepes(P : object):
    """ trie la pile P selon la méthode du tri crêpes """
    n = hauteur_pile(P)
    for i in range(n, 1, -1):
        # On recherche la plus grande crêpe.
        j = max_pile(P, i)
        # On retourne la pile à partir de cette crêpe pour mettre cette plus grande crêpe en haut.
        retourner(P, j)
        # On retourne cette partie pour que la plus grande crêpe se retrouve tout en bas.
        retourner(P, i)
```

Exercice 2

Q1.1

2

Q1.2

3 (droite) + 2 (bas) + 1 (départ) = 6

Q2

0, 0) → (0, 1) → (0, 2) → (0, 3) → (1, 3) → (2, 3) = 11

(0, 0) → (0, 1) → (0, 2) → (1, 2) → (1, 3) → (2, 3) = 10

(0, 0) → (0, 1) → (0, 2) → (1, 2) → (2, 2) → (2, 3) = 14

(0, 0) → (0, 1) → (1, 1) → (1, 2) → (1, 3) → (2, 3) = 9

(0, 0) → (0, 1) → (1, 1) → (1, 2) → (2, 2) → (2, 3) = 12

(0, 0) → (0, 1) → (1, 1) → (2, 1) → (2, 2) → (2, 3) = 12

(0, 0) → (1, 0) → (1, 1) → (1, 2) → (1, 3) → (2, 3) = 10

(0, 0) → (1, 0) → (1, 1) → (1, 2) → (2, 2) → (2, 3) = 14

(0, 0) → (1, 0) → (1, 1) → (2, 1) → (2, 2) → (2, 3) = 13

(0, 0) → (1, 0) → (2, 0) → (2, 1) → (2, 2) → (2, 3) = 16

soit 10 chemins au total

(0, 0) → (1, 0) → (2, 0) → (2, 1) → (2, 2) → (2, 3) = 4 + 2 + 3 + 2 + 5 + 1 = 16

Q3.1

4	5	6	9
6	6	8	10
9	10	15	16

Q3.2

Le déplacement sur la 1ère ligne correspond à i = 0 et 0 ≤ j ≤ 3.

La somme s'effectue avec la case précédente si elle existe (ie : j > 0).

Donc T'[0][j] = T[0][j] + T'[0][j-1] pour j > 0

Q4

On cherche le chemin maximum entre 2 chemins possibles qui viennent de la gauche (j-1) ou d'en haut (i-1).

Q5.1

On ne fait pas appel à la récurrence pour i = 0 et j = 0. Dans ce cas T[0][0] = 4 (cas de base).

Q5.2

```
def somme_max(T : list, i : int, j : int) -> int:
    if i == 0 and j == 0:
        return T[0][0]
    elif i == 0:
        return T[i][j] + somme_max(T, i, j-1)
    elif j == 0 :
        return T[i][j] + somme_max(T, i-1, j)
    else :
        return T[i][j] + max(somme_max(T, i-1, j), somme_max(T, i, j-1))
```

Q5.3

somme_max(T, len(T) - 1, len(T[0]) - 1))

Exercice 3

Q1

- Taille de l'arbre : nombre de nœuds = 9
- Hauteur de l'arbre : nœud le plus profond depuis la racine = 4

Q2.1

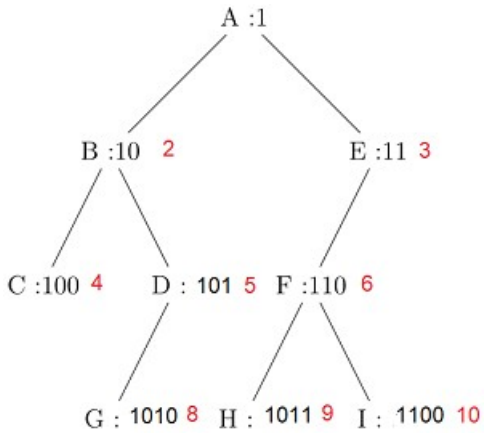
1010

Q2.2

I

Q2.3

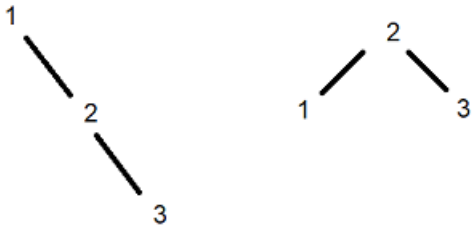
h bits



Q2.4

- Pour un arbre bien formé (complet) : $2^{h-1} - 1 < n \leq 2^h - 1$
- Pour un arbre complètement déséquilibré : n = h

Il vient : $h \leq n \leq 2^h - 1$



Q3.1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
15	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O

Q3.2

E(i/2)

Q4

```
def recherche(abr : list, elt : int) -> bool:
    """ recherche naïve d'un élément dans un abr sous forme de vecteur """
    for i in range(1, len(abr)):
        if elt == abr[i]:
            return True
    return False
```

Complexité O(n)

```
def recherche(abr : list, elt : int) -> bool:
    """ recherche optimisée d'un élément dans un abr sous forme de vecteur """
    i = 1
```

```
while i < len(abr):
    if elt == abr[i]:
        return True
    elif elt < abr[i]:
        i = 2*i
    else:
        i = 2*i + 1
return False
```

Complexité $O(\ln_2 n)$

Exercice 4

Q1.1

Une clef primaire permet de trouver de retrouver systématiquement les attributs d'une relation de façon sûre et unique.

Q1.2

La clef 133310FE (VARCHAR) a été transformée en 133310 (INT).

INSERT INTO seconde VALUES(133310, 'anglais', 'espagnol', '', '2A')

Q1.3

UPDATE seconde SET langue1 = 'allemand' WHERE num_eleve = 156929

Q2.1

Affiche tous les attributs num_eleve de la table seconde.

Q2.2

Affiche le nombre d'enregistrements de la table seconde.

Q2.3

SELECT COUNT(num_eleve) FROM seconde WHERE langue1 = 'allemand' OR langue2 = 'allemand'

Q3.1

La donnée n'a besoin d'être mise à jour qu'une seule fois dans une table.

Une entrée dans la table eleve n'est possible que si l'entrée existe au préalable dans la table seconde.

Q3.2

SELECT eleve.nom, eleve.prenom, eleve.datenaissance
FROM eleve, seconde
WHERE eleve.num_eleve = seconde.num_eleve AND seconde.classe = '2A'

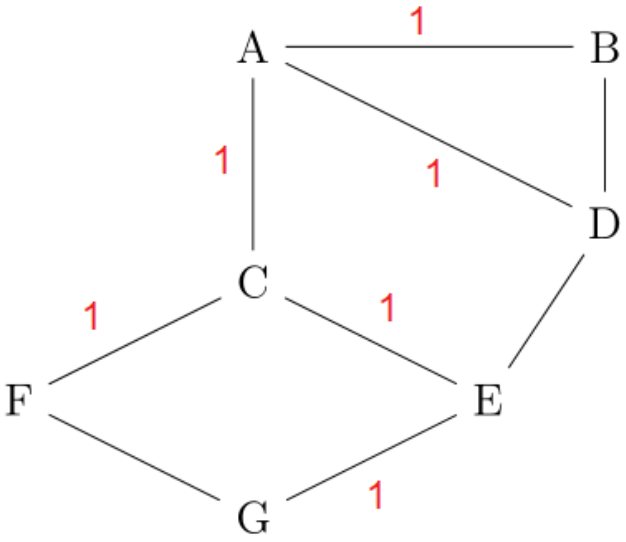
Q4

Table coordonnees	
Attributs	**Types**
idcoordonnees	INT (clef primaire)
num_eleve	INT (clef étrangère référence eleve)
adresse	VARCHAR
codepostale	VARCHAR
ville	VARCHAR
email	VARCHAR

Exercice 5

Q1.1

A → C → E → G



Q1.2

Table de routage du routeur G		
Destination	Routeur suivant	Distance
A	F	3
B	E	3
C	F	2
D	E	2
E	E	1
F	F	1

Q2

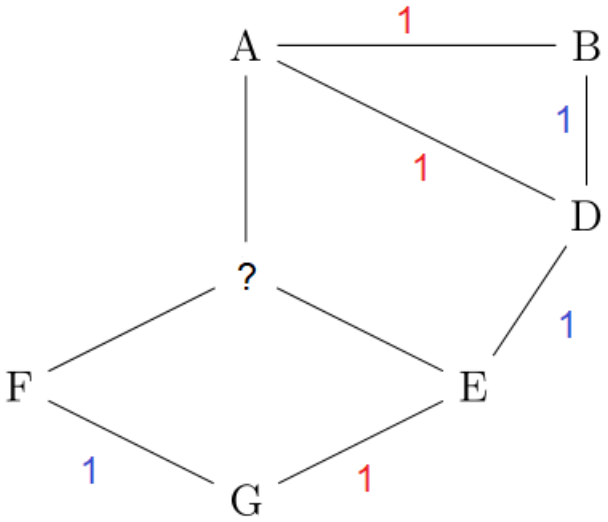


Table de routage du routeur A		
Destination	Routeur suivant	Distance
B	B	1
C	?	
D	D	1
E	D	2
F	D	4
G	D	3

Q3.1

Coût = 10^8 / d = 10^8 / (10 x 10^9) = 10^{-2} = 0,01

Q3.2

d = 10^8 / 5 = 2.10^7 = 20.10^6 = 20 Mb/s

Q4

On utilise l'algorithme de Dijsktra : on choisit le sommet non visité avec la distance la plus faible (le débit le plus élevé), puis on calcule la distance à travers lui à chaque voisin non visité, et on met à jour la distance du voisin si elle est plus petite.

On obtient alors le chemin suivant : A → D → E → G