

Exercice 1

1

8
5
2
4

2.1

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

2.2

```
def max_pile(P,i):
    # si la pile comporte moins de i élément ou que i=0 on renvoie 0
    if i > hauteur_pile(P) or i==0:
        return 0
    maxi = depiler(P)
    Q = creer_pile_vide()
    empiler(Q,maxi)
    j = 1
    indice = 1
    while j < i:
        j = j + 1
        x = depiler(P)
        if x > maxi:
            maxi = x
            indice = j
        empiler(Q,x)
    while not est_vide(Q):
        empiler(P, depiler(Q))
    return indice
```

3

```
def retourner(P,j):
    Q1 = creer_pile_vide()
    Q2 = creer_pile_vide()
    i = 0
    while not est_vide(P) and i < j:
        i = i + 1
        x = depiler(P)
        empiler(Q1, x)
    while not est_vide(Q1):
        x = depiler(Q1)
        empiler(Q2, x)
    while not est_vide(Q2):
        x = depiler(Q2)
        empiler(P, x)
```

4

```
def tri_crepes(P):
    N = hauteur_pile(P)
    i = N
    while i > 1:
        j = max_pile(P,i)
        retourner(P,j)
        retourner(P,i)
        i = i -1
```

Exercice 2

1.1

Le chemin comprend 2 déplacements vers le bas

1.2

Sachant que les déplacements en diagonale ne sont pas autorisés, il faudra obligatoirement se déplacer 3 fois vers la droite (parcours 4 cases) et 2 fois vers le bas (parcours 2 cases supplémentaires) quel que soit l'ordre de ces déplacements. On aura donc bien un chemin de longueur égale à 6 quel que soit le chemin emprunté.

2

avec le parcours 0,0 -> 1,0 -> 2,0 -> 2,1 -> 2,2 -> 2,3
on obtient la somme $4 + 2 + 3 + 1 + 5 + 1 = 16$ qui est la somme maximale.

3.1

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

3.2

La somme obtenue à la colonne j est égale à la somme obtenue à la colonne $j-1$ (à gauche de j) plus la valeur de la case $0, j$ (puisque l'on peut uniquement aller à droite)

$$d'où T'[0][j] = T[0][j] + T'[0][j-1]$$

4

Quand on se trouve à la case (i, j) , on vient soit de la case $(i-1, j)$ (case située au-dessus de (i, j)), soit de la case $(i, j-1)$ (case située à gauche de (i, j)). Donc on doit ajouter à la valeur de la case $T[i][j]$ soit la somme obtenue à la case $(i-1, j)$, soit la somme obtenue à la case $(i, j-1)$ (on prendra la somme maximum).

$$d'où : T'[i][j] = T[i][j] + \max(T'[i-1][j], T'[i][j-1])$$

5.1

Le cas de base est le cas où $i = 0$ et $j = 0$, on renvoie alors la valeur $T[0][0]$

5.2

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

5.3

Pour résoudre le problème initial, on doit effectuer l'appel suivant : `somme_max(T, 2, 3)`

Exercice 3

1

taille = 9 ; hauteur = 4

2.1

G : 1010

2.2

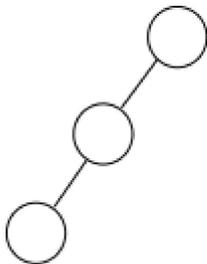
noeud l

2.3

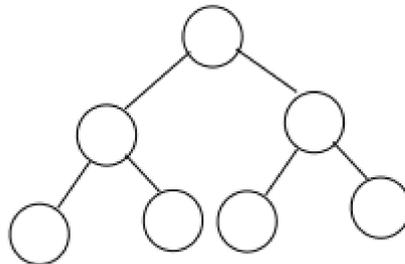
À chaque "étage", on augmente le nombre de bits de 1 : si $h = 1$, nombre de bit = 1 ; si $h = 2$, nombre de bits = 2... pour une hauteur h le nombre de bits est de h .

2.4

Prenons un exemple avec $h = 3$: nous avons 2 cas extrêmes : un arbre filiforme ou un arbre complet. Toutes les autres possibilités sont des cas intermédiaires.



arbre filiforme



arbre complet

Dans le cas de l'arbre filiforme nous avons, pour $h = 3$, $n = 3$ (n : taille). Si on généralise pour un arbre de hauteur h , nous avons $n = h$

Dans le cas d'un arbre complet, pour $h = 3$ nous avons $n = 7$, donc $n = 2^3 - 1 = 7$. Si on généralise pour un arbre de hauteur h , nous avons $n = 2^h - 1$

Sachant qu'un arbre quelconque est un intermédiaire entre l'arbre filiforme et l'arbre complet, nous pouvons donc dire que :

$$h \leq n \leq 2^h - 1$$

3.1

[15, 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O']

3.2

$\frac{i}{2}$ si i est pair et $\frac{i-1}{2}$ si i est impair

4

```
def recherche(arbre, element):
    n = arbre[0]
    i = 1
    while i <= n:
        if arbre[i] == element:
            return True
        elif element > arbre[i]:
            i = 2*i+1
        else :
            i = 2*i
    return False
```

Exercice 4

ATTENTION : Il y a une erreur dans l'énoncé : num_eleve ne peut pas être en entier car il est de la forme 133310FE (ou alors, c'est un entier en base 16 ;-))

1.1

num_eleve va jouer le rôle de clé primaire. La clé primaire permet d'identifier de manière unique un t-uplet de la relation seconde.

1.2

```
INSERT INTO seconde
(num_eleve, langue1, langue2, option, classe)
VALUES
('133310FE', 'anglais', 'espagnol', '0', '2A')
```

on a choisi de mettre 0 pour l'option quand l'élève n'a pas d'option

1.3

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

2.1

Cette requête donne tous les num_eleve contenus dans la table seconde

2.2

Le résultat est 30 car il y a 30 entrées

2.3

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

3.1

La clé étrangère permet de créer un lien entre la table *eleve* et la table *seconde*. Ce lien est unique pour chaque entrée. Pour préserver l'intégrité de la base de données : pour que toutes les valeurs de la clé étrangère correspondent bien à des valeurs présentes dans la clé primaire.

3.2

```
SELECT nom, prenom, datenaissance
FROM eleve
INNER JOIN seconde ON eleve.num_eleve = seconde.num_eleve
WHERE classe = '2A'
```

4

coordonnees
num_eleve (clé primaire, clé étrangère des tables <i>seconde</i> et <i>eleve</i>)
adresse
code_postal
ville
adresse_email

Exercice 5

1.1

A -> C -> F -> G

1.2

Destination	Routeur suivant	Distance
A	F	3
B	E	3
C	F	2
D	E	2
E	E	1
F	F	1

2

Destination	Routeur suivant	Distance
B	B	1
D	D	1
E	D	2
F	D	4
G	D	3

3.1

A -> B : 10 Gb/s soit le coût = $\frac{10^8}{10 \cdot 10^9} = 0,01$

3.2

$$\frac{10^8}{d} = 5 \Rightarrow d = \frac{10^8}{5} = 2 \cdot 10^7 \text{ b/s} = 20 \text{ Mb/s}$$

4

On part de A (possible d'aller en B, en D ou en C), on trouve le débit le plus important au niveau de la liaison A->D. Une fois en D, on va vers E et une fois en E, on rejoint directement G (car le coût du chemin E -> C -> F -> G serait plus grand).

d'où le chemin : A -> D -> E -> G avec un coût = $\frac{10^8}{10 \cdot 10^9} + \frac{10^8}{100 \cdot 10^9} + \frac{10^8}{100 \cdot 10^6} = 1,01$