Proposition de correction
Sujet Zero Ecrit Terminale NSI

Exercice 1

Question 1
Au début P = 4 2 5 8
1er passage : On dépile P, c'est-à-dire on récupère le sommet 4 et on met 4 dans Q
On a donc :
P= 2 5 8 et Q = 4 (sommet = élément le plus à gauche ici pour commodité écriture)

2ème passage : On dépile P, c'est-à-dire on récupère le sommet 2 et on met 2 dans Q
On a donc :
P= 5 8 et Q = 2 4 (2 est le sommet de la pile Q)

3ème passage : On dépile P, c'est-à-dire on récupère le sommet 5 et on met 5 dans Q
On a donc :
P= 8 et Q = 5 2 4 (5 est le sommet de la pile Q)

3ème passage : On dépile P, c'est-à-dire on récupère le sommet 8 et on met 8 dans Q
On a donc :
P= null et Q = 8 5 2 4 (8 est le sommet de la pile Q)

Question 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
```

Code à rajouter. On compte le nombre de dépilements

La pile étant sauvegardé dans Q, il suffit ensuite de la récupérer en dépilant Q (P ne doit pas être modifié à la sortie de la fonction)

2- Fonction max_pile
L'algo : L'idée centrale qu'il faut voir c'est qu'on doit dépiler p , sauvegarder dans une variable X . Ensuite on compare X à maxi. Si X> maxi alors maxi = X.
Pensez à expliciter votre algo avant d'écrire votre programme, cela vous permettra de récupérer des points à défaut d'avoir un programme juste …

```python
def max_pile(P, i):
    assert i <= hauteur_pile(P), "La pile compte moins de i éléments"

    # Initialisation
    rang = 1  # L'indice de l'élément en cours de traitement
    rang_maxi = 1  # le rang du maximum en cours
    maxi = depiler(P)  # au début, le maximum est le premier élément ...
    empiler(P, maxi)  # ... que l'on rempile immédiatement
    Q = creer_pile_vide()  # une pile vide pour stocker les éléments traités

    # On lit tous les éléments jusqu'au i-ième pour trouver le maximum
    while rang <= i:
        x = depiler(P)
        if x > maxi:
            maxi = x
            rang_maxi = rang
        empiler(Q, x)
        rang += 1

    # On reconstitue la pile P
    while not(est_vide(Q)):
        empiler(P, depiler(Q))

    return rang_maxi
```

La pile ne devant pas être modifiée, on la reconstitue avant de sortir de la fonction

<u>Question 3</u>
<u>Proposition d'algo :</u>
<u>1ère étape :</u>

4		
2		
5		
8		
Pile P	Pile Q	Pile R

J= 3. Il faut dépiler P et empiler Q j fois.

2		
5		
8	4	
Pile P	Pile Q	Pile R

Dépile P et empile Q. J=2

5	2	
8	4	
Pile P	Pile Q	Pile R

Dépile P et empile Q. J= 1

	5	
	2	
8	4	
Pile P	Pile Q	Pile R

Dépile P et empile Q . j=0 Terminé.

2ème Etape :

	2	
8	4	5
Pile P	Pile Q	Pile R

Dépile Q et empile R jusqu'à Q vide.

		2
8	4	5
Pile P	Pile Q	Pile R

Dépile Q et empile R

		4
		2
8		5
Pile P	Pile Q	Pile R

Dépile Q et empile R . Arrêt car Q vide

3ème Etape :

Dépiler R et rempiler P

4		2
8		5
Pile P	Pile Q	Pile R

Dépile R et empiler P jusqu'à R vide

2		
4		
8		5
Pile P	Pile Q	Pile R

Dépile R et empiler P jusqu'à R vide

5		
2		
4		
8		
Pile P	Pile Q	Pile R

Dépile R et empiler P Arrêt R est Vide

Proposition de codage :

```python
def retourner(P, j):
    assert j <= hauteur_pile(P), "La pile compte moins de j éléments"

    # Initialisation
    Q = creer_pile_vide()  # une pile vide pour vider P
    R = creer_pile_vide()  # une pile vide pour vider Q
    rang = 1  # le rang de l'élément en cours de traitement

    # On dépile les j premiers éléments dans Q
    while rang <= j:
        empiler(Q, depiler(P))
        rang += 1

    # On vide Q dans R
    while not(est_vide(Q)):
        empiler(R, depiler(Q))

    # On vide R dans P
    while not(est_vide(R)):
        empiler(P, depiler(R))

    # La fonction ne renvoie rien (en réalité None en python)
    # On peut tout aussi bien se passer de retour
    # ce qui aura le même effet lors de l'exécution
    return None
```

Question 4

```
Fonction tri_crepes(P) :
h = hauteur_pile(P)
Pour i allant de 0 à h-1 (exclus) :
        # inutile de retourner la dernière crêpe en fin de boucle car c'est la plus petite
        rang_maxi = maxi_pile(P,h-i)
        # on cherche le rang de l'élément maximal parmi les h-i premiers
        retourner(P,rang_maxi)      # On retourne le haut de la pile jusqu'à l'élément
        maximal
        retourner(P,h-i)            # On retourne toute la pile jusqu'en bas
```

```python
def tri_crepes(P) :
    assert not est_vide(P), "Il n'y a pas de crêpes à trier !'

    h = hauteur_pile(P)

    for i in range(0, h-1):
        rang_maxi = max_pile(P, h-i)
        retourner(P, rang_maxi)
        retourner(P, h-i)
```

EXERCICE 2

Question 1

1. Pour aller de la case (0, 0) à la case (2, 3) on fait 3 déplacements vers la droite et 2 vers le bas.
2. Comme on fait des déplacements de 1 pas à chaque étape, il faut faire 2 + 3 = 5 déplacements. Chaque déplacement nous amène sur une nouvelle case. En n'oubliant pas d'inclure la case (0, 0) il faut donc parcourir 2 + 3 + 1 = 6 cases.

Question 2

-Les différents parcours possibles sachant qu'on ne peut que descendre ou aller à droite :

4	**1**	**1**	**3**
2	0	2	**1**
3	1	5	**1**

Somme = 11

4	**1**	**1**	3
2	0	**2**	**1**
3	1	5	**1**

Somme = 10

4	**1**	**1**	3
2	0	**2**	1
3	1	**5**	**1**

Somme = 14

4	**1**	1	3
2	**0**	**2**	**1**
3	1	5	**1**

Somme = 9

4	**1**	1	3
2	**0**	**2**	1
3	1	**5**	**1**

Somme = 13

4	**1**	1	3
2	**0**	2	1
3	**1**	**5**	**1**

Somme = 12

4	1	1	3
2	**0**	**2**	**1**
3	1	5	**1**

Somme = 10

4	1	1	3
2	**0**	**2**	1
3	1	**5**	**1**

Somme = 14

4	1	1	3
2	**0**	2	1
3	**1**	**5**	**1**

Somme = 13

4	1	1	3
2	0	2	1
3	**1**	**5**	**1**

Somme = 16

La somme max est donc de 16

Question 3

1-

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

Tableau des valeurs **MAX**

2- Justifier que si j est différent de 0, alors : T' [0] [j] = T [0] [j] + T' [0] [j-1]

La valeur T'[0][j] où j est non nul correspond à la somme des cases (0, 0) à (0, *j*), c'est à dire des cases de la première ligne du tableau.

Il n'y a qu'un seul chemin qui corresponde à cette somme et il passe obligatoirement par la case à gauche (d'indice j-1) de la case $(0, j)$.
Donc pour calculer la somme T'[0][j] on ajoute simplement la valeur de la case $(0, j)$ (c'est à direT[0][j]) à la somme obtenue à la case précédente (c'est à dire T'[0][j-1]).

On a donc bien T'[0][j] = T[0][j]+T'[0][j-1].

Question 4

Justifier que si i et j sont différents de 0, alors : T' [i] [j] = T [i] [j] + max(T' [i-1] [j], T' [i] [j-1]).

Si i et j son non-nuls, il y a deux chemins amenant à la case (i, j). Le premier provient de la case du dessus $(i-1, j)$, le second de la case de gauche $(i, j-1)$.
La valeur de T'[i][j] s'obtient donc en ajoutant la valeur de T[i][j] au maximum des deux chemins menant à cette case : max(T'[i-1][j],T'[i][j-1]).

Question 5

1. Le cas de base est atteint lorsque l'on atteint une case de la première ligne (i vaut 0) ou de la première colonne (j vaut 0). Dans ce cas on calcule la somme en additionnant la valeur de la case en question avec le résultat de somme_max avec comme argument T et la case précédente (sur la ligne si i=0 ou la colonne si j=0).

2-

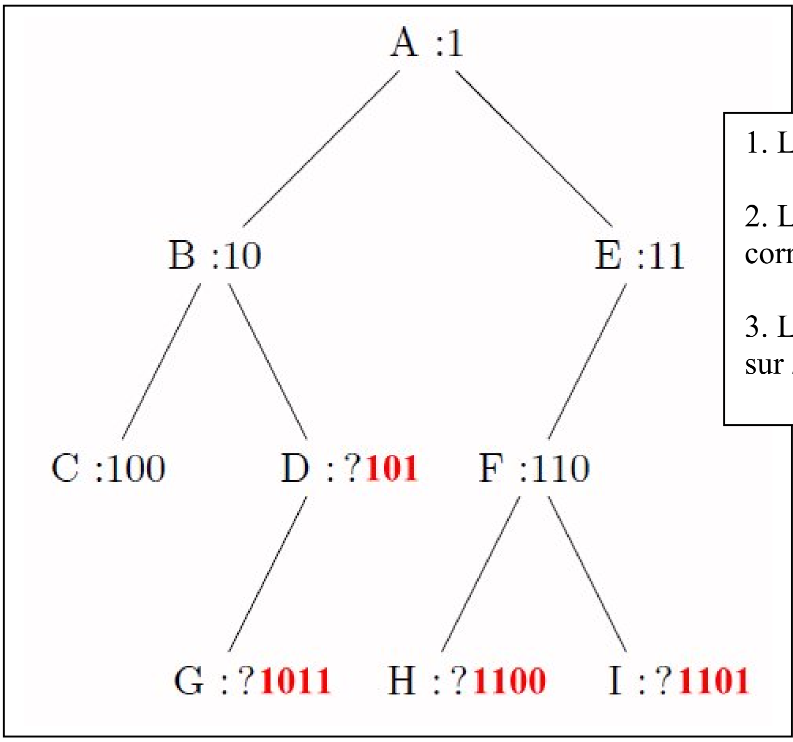
```
def somme_max(T, i, j):
if i == 0 and j == 0:
    return T[0][0]
elif i == 0 :
    return T[0][j] + somme_max(T,0,j-1)
elif j == 0 :
    return T[i][0] + somme_max(T,i-1,0)
else :
    return T[i][j] + max(somme_max(T,i-1,j), somme_max(T,i,j-1))
```

3- On appelle somme_max(T,2,3)

Exercice 3

Question 1. La taille d'un arbre correspond au nombre de noeuds. Ici elle vaut 9. La hauteur de l'arbre est la longueur du chemin le plus long entre la racine et l'une des feuilles. Ici 4.

Question 2

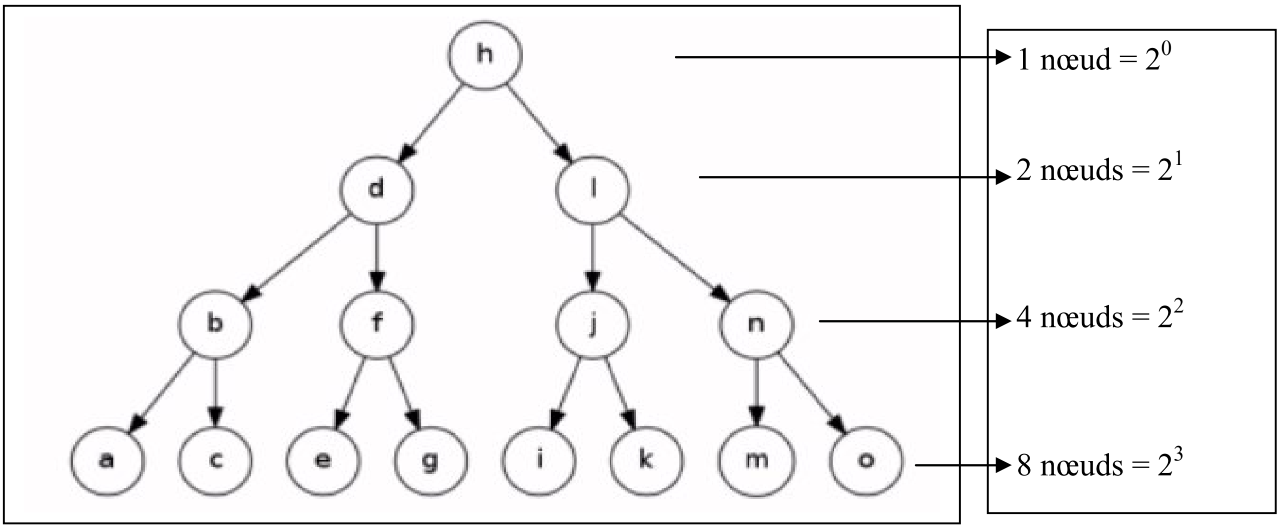


1. Le numéro associé à *G* est 1010.

2. Le nombre 13 s'écrit 1101 en binaire. Il correspond donc au noeud *I*.

3. Les noeuds les plus bas sont numérotés sur *h* bits (4 dans l'exemple).

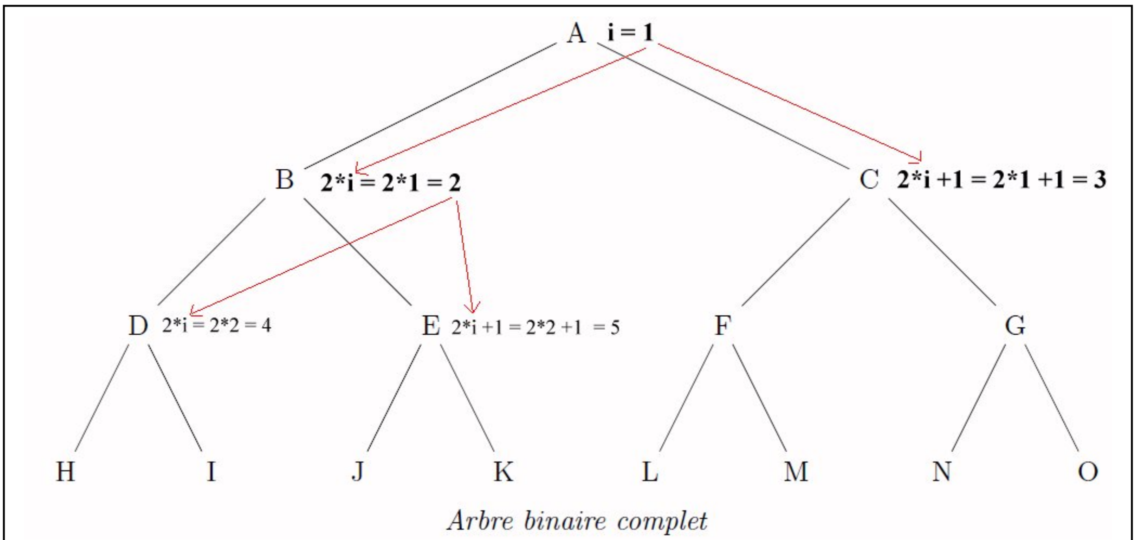
4- Pour justifier encadrement, voyons un exemple (Note : c'est pas une démonstration …)



Pour h = 4, nombre de nœuds total = 1+2+4+8 = 15 = $2^4 - 1$
En généralisant, le nombre de nœuds total si arbre complet = $2^h - 1$

Conclusion, si n est la taille de l'arbre, on a bien $h \leqslant n \leqslant 2^h - 1$

Question 3



Arbre binaire complet

Indice i : 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

1. Le tableau est : [None,"A","B","C","D","E","F","G","H","I","J","K","L","M","N","O"]

2. Voyons par exemple le nœud D, indice 4. Son père est B, indice 2 , soit 4/2 = 2
Le nœud E , indice 5 a aussi B comme père indice 2 soit 5/2 = 2.5 et on veut 2. Il suffit donc de prendre la partie entière de la division, c-a-d i//2 en python.
Donc on prend l'indice du fils gauche ou droit, on divise par 2 et on prend la partie entière pour obtenir l'indice du père.

Question 4

```
def recherche(arbre, element):
    taille = len(arbre)

    i = 1
    while i < taille:
        if arbre[i] == element:
            return True
        elif element < arbre[i]:
            i = 2*i
        else:
            i = 2*i+1

    return False
```

Exercice 4

Question 1

1. L'attribut num_eleve est la clé primaire. Elle permet d'identifier de façon unique chacun des enregistrements de la table (relation)

2. Pour ajouter l'élève ACHIR Mussa à la table seconde on écrit :

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

3. Pour changer la langue de l'élève altmeyer , on fait :

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

Question 2

1. Cette requête renvoie les numéros d'identification de tous les élèves de seconde. Il s'agit donc des données de la première "colonne" du fichier csv.

2. Cette requête permet de compter le nombre d'élèves de seconde, soit 30 ici

3- Pour connaître le nombre d'élèves qui font allemand en langue 1 ou langue 2 :

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

Question 3

1. L'ajout d'une clé étrangère permet de s'assurer que les données des tables se correspondent. Elle peut aussi permettre d'empêcher d'ajouter des objets dans une table s'ils ne sont pas présents dans l'autre.

2. On fait :

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

Autre possibilité :

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

Question 4

Nom de la table : coordonnees

Structure :

num_eleve	type varchar	clé primaire	clé étrangère de la table seconde
adresse	type varchar		
code postal	int		
ville	type varchar		
mail	type varchar		

Exercice 5

Question 1
Sur le schéma , il est facile de déterminer le(s) chemin(s) les plus court(s).
Via table de routage : Une valeur de 1 au niveau de la distance indique ici que le routeur est accessible directement. On va donc rechercher des passages d'une table à l'autre en passant par une distance de 1. Si 2 chemins de distance minimale, on garde le 1er trouvé.

1- En lisant la table de routage de A puis celles de C et F on obtient A → C → F → G. (Autre possibilité : A C E G)

2- Table de routage du routeur G , protocole RIP :

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

Question 2
Routeur C en panne, déterminer Table routage de A suivant RIP

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

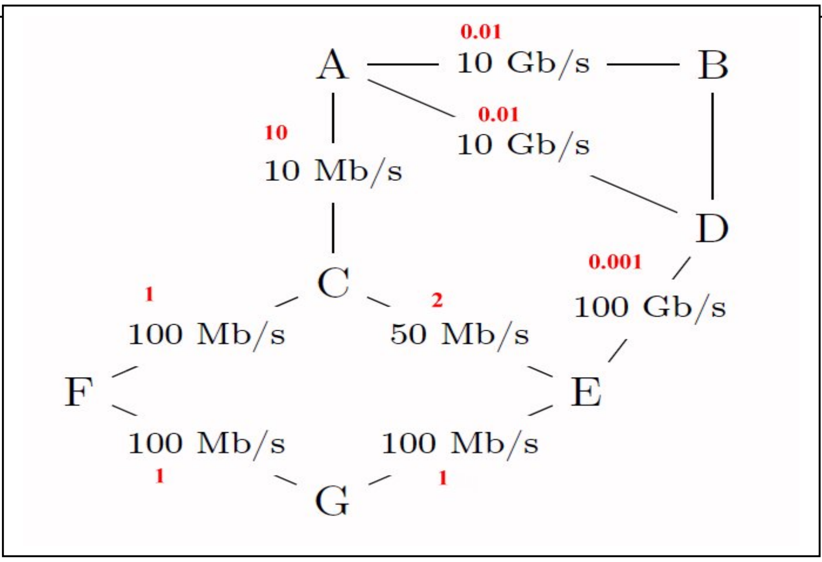
Question 3

1. La liaison entre A et B a un débit de 10 Gb/s = $10 * 10^9$ b/s.

Donc son coût vaut $10^8 / 10 * 10^9 = 0.01$

2. Si le coût est de 5 alors on a $10^8 * d = 5$ ce qui donne $d = 10^8 / 5 = 20.10^6$ b/s. = 20M b/s

Question 4



On déduit que le coût le plus faible est A D E G.
0.01+0.001+1=1.011