

Centres étrangers 2023 Jour 2

Exercice 1

Partie A : L'adressage IP

1.
 - a. Toute adresse IP du type 192.168.5.XYZ avec XYZ différente de 000, 255 et 003 est valide pour le routeur F. En effet, l'adresse 192.168.5.0 est celle du réseau lui-même, l'adresse 192.168.5.255 correspond en général à l'adresse de diffusion et l'adresse 192.168.5.3 est déjà utilisée par une machine. On peut donc par exemple affecter l'adresse 192.168.5.1 au routeur F.
 - b. En tenant compte des remarques précédentes, XYZ peut prendre les valeurs de 1 à 254. Il y a donc 254 adresses IP valides pour le réseau F.
2.
 - a. Le masque de sous-réseau du réseau B est 255.255.240.0.
 - b. Une des machine du réseau B a pour adresse IP 192.168.2.2. Pour déterminer le masque de sous-réseau, on convertit ces adresses en binaire. On obtient alors 11000000.10101000.00000010.00000010 pour la machine et 11111111.11111111.11110000.00000000 pour le masque. En effectuant un ET logique bit à bit, on obtient 11000000.10101000.00000000.00000000 qui correspond à l'adresse du réseau B. On peut donc conclure que le masque de sous-réseau du réseau B est 192.168.0.0.
 - c. L'interconnexion entre les routeurs A, B, E et F permet, en cas de défaillance de l'un d'entre eux, de maintenir la liaison entre toutes les machines représentées sur le schéma.

Partie B : Le routage

1.
 - a. Il existe un chemin de longueur 2 entre le routeur A et le routeur E : A - B - E. Il s'agit du plus courts chemin possible en terme de nombre de sauts. Pour aller de F vers B, il existe plusieurs chemins optimaux en termes de nombre de sauts. Ce sont tous les chemins de longueur 3 : F - D - A - B, F - H - G - B et F - H - E - B.

Tableau 1: Table de routage du routeur E

Destination	Routeur suivant	Distance
A	B	2
B	B	1
C	H	2
D	G	2
E	E	0
F	H	2
G	G	1
H	H	1

Tableau 2: Table de routage du routeur G

Destination	Routeur suivant	Distance
A	B	2
B	B	1
C	D	1
D	D	1
E	E	1
F	D	2
G	G	0
H	H	1

b.

Tableau 3: Table de routage du routeur F

Destination	Routeur suivant	Coût total
A	D	1.1
B	H	10.11
C	D	1.1
D	D	0.1
E	H	10.1
G	D	1.1
H	H	0.1

2. a.

b. Entre le routeur E et le routeur D, le chemin optimal est E - H - F - D, dont le coût total est de 10.2.

Exercice 2

1. a. Le résultat de la requête est le suivant :

age	taille	poids
6	1.70	100

- b. La requête est la suivante :

```
SELECT nom, age
FROM animal
WHERE nom_espece = 'bonobo'
ORDER BY age
```

2. a. L'attribut `nom_espece` peut vraisemblablement servir de clé primaire pour la relation `espece` car deux espèces différentes doivent avoir des noms différents. L'attribut `num_enclos` est une clé étrangère relative à la clé primaire `num_enclos` de la relation `enclos`.

- b. Schéma relationnel de la base de données :

- animal(id_animal : INT, nom : VARCHAR, age : INT, taille : FLOAT, poids : INT, #nom_espece : VARCHAR)
- enclos(num_enclos : INT, ecosysteme : VARCHAR, surface : INT, struct : VARCHAR, date_entretien : DATE)
- espece(nom_espece : VARCHAR, classe : VARCHAR, alimentation : VARCHAR, #num_enclos : INT)

3. a. La requête suivante corrige l'erreur signalée :

```
UPDATE espece
SET classe='mammifères'
WHERE nom_espece='ornithorynque'
```

- b. La requête suivante permet d'intégrer le nouveau venu dans la base de données :

```
INSERT INTO animal VALUES (179, 'Serge', 0, 0.8, 30, 'lama')
```

4. a. Requête permettant de recenser le nom et l'espèce de tous les animaux carnivores vivant en vivarium dans le zoo :

```
SELECT nom, nom_espece
FROM animal
```

```
JOIN espece ON animal.nom_espece = espece.nom_espece
JOIN enclos ON espece.num_enclos = enclos.num_enclos
WHERE enclos.struc = 'vivarium' and espece.alimentation = 'carnivore'
```

- b. Requête permettant de connaître le nombre d'oiseaux dans tout le zoo :

```
SELECT COUNT(*)
FROM animal
JOIN espece ON animal.nom_espece = espece.nom_espece
WHERE espece.classe = 'oiseau'
```

Exercice 3

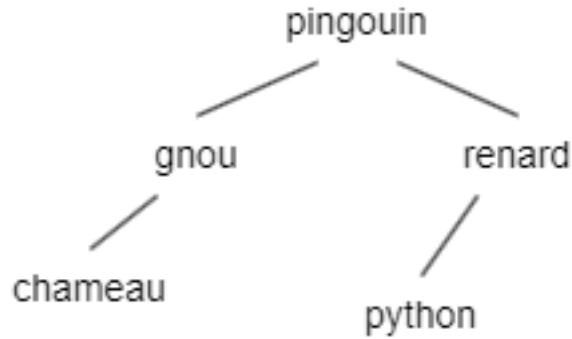
1.
 - a. La fonction retourne : `Bonjour Alan !`.
 - b. `x` et `y` sont deux variable booléennes. `x` est la valeur de vérité de la comparaison entre les caractères `n` et `j`, elle prend donc la valeur `False`. `y` est la valeur de vérité de la comparaison entre les caractères `o` et `o`, elle prend donc la valeur `True`.
 - c. La fonction suivante prend en paramètre une chaîne `une_chaine` et une lettre `une_lettre` et retourne le nombre de fois où la lettre `une_lettre` apparaît dans la chaîne `une_chaine` :

```
def occurrences_lettre(une_chaine, une_lettre):
    """Retourne le nombre d'occurrences de la lettre une_lettre dans la chaîne une_chaine"""
    compteur = 0
    for lettre in une_chaine:
        if lettre == une_lettre:
            compteur += 1
    return compteur
```

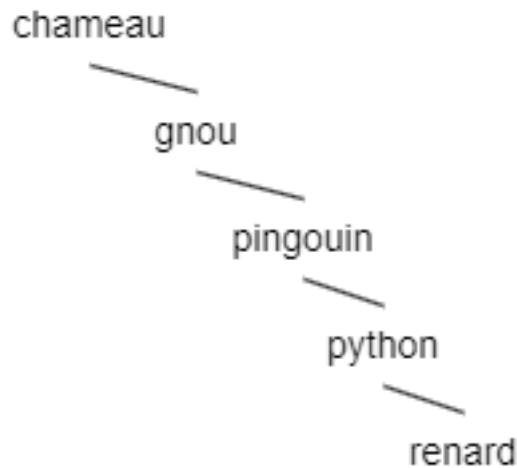
2.
 - a. Pour obtenir un arbre binaire de hauteur minimale, on range les mots dans l'ordre alphabétique et on place le mot du milieu à la racine. On répète l'opération sur les deux sous-arbres de gauche et de droite.

Liste dans l'ordre alphabétique : `['chameau', 'gnou', 'pingouin', 'python', 'renard']`.

On obtient l'arbre :



- b. Pour obtenir un arbre de hauteur maximale, on peut placer à la racine le premier mot de la liste classée dans l'ordre alphabétique, puis placer le mot suivant en sous-arbre droit et chaque mot suivant en sous-arbre droit du précédent. On obtient un arbre filiforme.



3. a. `mystere(abr_mots_francais)` retourne 336 531. Cette fonction calcule en effet de façon récursive le nombre d'éléments de l'arbre binaire donné en paramètre, égal à un (on compte la racine, si l'arbre n'est pas vide) plus le nombre d'éléments de l'arbre binaire de gauche plus le nombre d'éléments de l'arbre binaire de droite.
- b. Fonction permettant de calculer la hauteur d'un arbre binaire :

```

def hauteur(un_abr):
    """Retourne la hauteur de l'arbre binaire un_abr."""
    if un_abr.est_vide():
        return 0
    else:

```

```
        return 1 + max(hauteur(un_abr.sous_arbre_gauche), hauteur(un_abr.sous_arbre_d
```

4. a. Code de la fonction complétée :

```
def chercher_mots(liste_mots, longueur, lettre, position):  
    res = []  
    for i in range(len(liste_mots)):  
        if len(liste_mots[i]) == longueur and liste_mots[i][position] == lettre:  
            res.append(liste_mots[i])  
    return res
```

- b. La commande `chercher_mots(liste_mots_francais, 3, 'x', 2)` retourne la liste des mots français de longueur 3 contenant la lettre x à la troisième position. La commande `chercher_mots(chercher_mots(liste_mots_francais, 3, 'x', 2), 3, 'a', 1)` retourne, parmi ceux-ci, les mots qui possèdent un 'a' à la deuxième position, soit à partir de l'exemple donné dans l'énoncé : ['fax', 'max'].

- c. Code permettant de trouver les mots de 5 lettres qui se terminent par 'ter' :

```
chercher_mots(chercher_mots(chercher_mots(liste_mots_francais, 5, 't', 2), 5, 'e', 3)
```