

Bac blanc 2022 – Correction

EXERCICE 1 : *Cet exercice porte sur les réseaux et les protocoles de routage.*

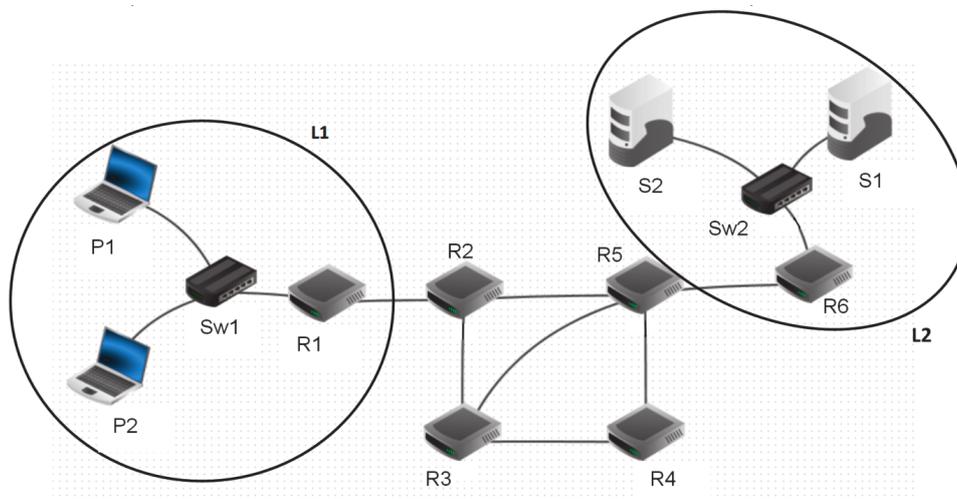


Figure 1 : Réseau d'entreprise

La figure 1 ci-dessus représente le schéma d'un réseau d'entreprise. Il y figure deux réseaux locaux L1 et L2. Ces deux réseaux locaux sont interconnectés par les routeurs R2, R3, R4 et R5. Le réseau local L1 est constitué des PC portables P1 et P2 connectés à la passerelle R1 par le switch Sw1. Les serveurs S1 et S2 sont connectés à la passerelle R6 par le switch Sw2. Le tableau 1 suivant indique les adresses IPv4 des machines constituant le réseau de l'entreprise.

Nom	Type	Adresse IPv4
R1	routeur (passerelle)	Interface 1 : 192.168.1.1/24 Interface 2 : 10.1.1.2/24
R2	routeur	Interface 1 : 10.1.1.1/24 Interface 2 : 10.1.2.1/24 Interface 3 : 10.1.3.1/24
R3	routeur	Interface 1 : 10.1.2.2/24 Interface 2 : 10.1.4.2/24 Interface 3 : 10.1.5.2/24
R4	routeur	Interface 1 : 10.1.5.1/24 Interface 2 : 10.1.6.1/24
R5	routeur	Interface 1 : 10.1.3.2/24 Interface 2 : 10.1.4.1/24 Interface 3 : 10.1.6.2/24 Interface 4 : 10.1.7.1/24
R6	routeur (passerelle)	Interface 1 : 172.16.0.1/16 Interface 2 : 10.1.7.2/24
P1	ordinateur portable	192.168.1.40/24
P2	ordinateur portable	192.168.1.46/24
S1	serveur	172.16.8.10/16
S2	serveur	172.16.9.12/16

Tableau 1 : adresses IPv4 des machines

Rappels et notations

Rappelons qu'une adresse IP est composée de 4 octets, soit 32 bits. Elle est notée X1.X2.X3.X4, où X1, X2, X3 et X4 sont les valeurs des 4 octets. Dans le tableau 1, les valeurs des 4 octets ont été converties en notation décimale.

La notation X1.X2.X3.X4/n signifie que les n premiers bits de poids forts de l'adresse IP représentent la partie « réseau », les bits suivants de poids faibles représentent la partie « machine ».

Toutes les adresses des machines connectées à un réseau local ont la même partie réseau. L'adresse IP dont tous les bits de la partie « machine » sont à 0 est appelée « adresse du réseau ». L'adresse IP dont tous les bits de la partie « machine » sont à 1 est appelée « adresse de diffusion ».

1) a) Quelles sont les adresses des réseaux locaux L1 et L2?

Solution : 192.168.1.0/24 pour L1 et 172.16.0.0/16 pour L2.

b) Donner la plus petite et la plus grande adresse IP valides pouvant être attribuées à un ordinateur portable ou un serveur sur chacun des réseaux L1 et L2 sachant que l'adresse du réseau et l'adresse de diffusion ne peuvent pas être attribuées à une machine.

Solution : 192.168.1.1 à 192.168.254 pour L1 et 172.16.0.1 à 172.16.255.254 pour L2.

c) Combien de machines peut-on connecter au maximum à chacun des réseaux locaux L1 et L2? On donne ci-dessous les valeurs de quelques puissances de 2?

Solution : 254 pour L1 et 65534 pour L2.

2^6	2^7	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}
64	128	256	512	1024	2048	4096	8192	16384	32768	65536	131072

2) a) Expliquer l'utilité d'avoir plusieurs chemins possibles reliant les réseaux L1 et L2.

Solution : Si un routeur tombe en panne ou est saturé, on peut utiliser une autre route.

b) Quel est le chemin le plus court en nombre de sauts pour relier R1 et R6? Donner le nombre de sauts de ce chemin et préciser les routeurs utilisés.

Solution : C'est R1-R2-R5-R6, qui nécessite 3 sauts.

c) La bande passante d'une liaison Ether (quantité d'information qui peut être transmise en bits/s) est de 10^7 bits/s et celle d'une liaison FastEther est de 10^8 bits/s. Le coût d'une liaison est défini par $10^8/d$, où d est sa bande passante en bits/s.

Liaison	R1-R2	R2-R5	R5-R6	R2-R3	R3-R4	R4-R5	R3-R5
Type	Ether	Ether	Ether	FastEther	FastEther	FastEther	Ether

Tableau 2 : type des liaisons entre les routeurs

Quel est le chemin reliant R1 et R6 qui a le plus petit coût? Donner le coût de ce chemin et préciser les routeurs utilisés.

Solution : On trouve le tableau :

Liaison	R1-R2	R2-R5	R5-R6	R2-R3	R3-R4	R4-R5	R3-R5
Type	10	10	10	1	1	1	10

Le chemin le moins coûteux est R1-R2-R3-R4-R5-R6 qui a un coût de 23.

3) On donne ci-dessous les tables de routage des routeurs R1, R2, R5 et R6 au démarrage du réseau. Compléter les lignes laissées vides des tables de routage des routeurs R5 et R6 pour que les échanges entre les ordinateurs des réseaux L1 et L2 se fassent en empruntant le chemin le plus court en nombre de sauts.

Table de routage de R1 :

IP Réseau de destination	Passerelle	Interface
192.168.1.0/24	–	Interface 1
10.1.1.0/24	–	Interface 2
0.0.0.0/0	10.1.1.1	Interface 2

Table de routage de R2 :

IP Réseau de destination	Passerelle	Interface
10.1.1.0/24	–	Interface 1
10.1.2.0/24	–	Interface 2
10.1.3.0/24	–	Interface 3
192.168.1.0/24	10.1.1.2	Interface 1
172.16.0.0/16	10.1.3.2	Interface 3

Table de routage de R5 :

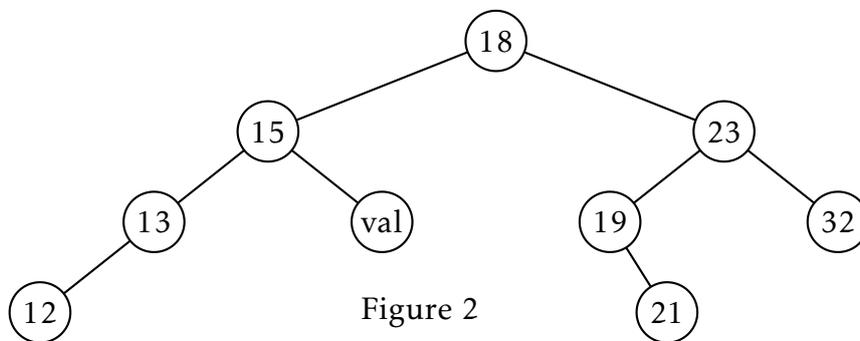
IP Réseau de destination	Passerelle	Interface
10.1.3.0/24	–	Interface 1
10.1.4.0/24	–	Interface 2
10.1.6.0/24	–	Interface 3
10.1.7.0/24	–	Interface 4
172.16.0.0/16	10.1.7.2	Interface 4
192.168.1.0/24	10.1.3.1	Interface 1

Table de routage de R6 :

IP Réseau de destination	Passerelle	Interface
172.16.0.0/16	–	Interface 1
10.1.7.0/24	–	Interface 2
192.168.1.0/24	10.1.7.1	Interface 2

EXERCICE 2 : *Cet exercice porte sur les arbres binaires de recherche.*

Dans cet exercice, les arbres binaires de recherche ne peuvent pas comporter plusieurs fois la même clé. De plus, un arbre binaire de recherche limité à un nœud a une hauteur de 1. On considère l'arbre binaire de recherche représenté ci-dessous (figure 2), où "val" représente un entier :



1) a) Donner le nombre de feuilles de cet arbre et préciser leur valeur (étiquette).

Solution : Il y a 4 feuilles qui sont 12, val, 21 et 32.

b) Donner le sous-arbre gauche du nœud 23.

Solution : C'est l'arbre ci-contre.



c) Donner la hauteur et la taille de l'arbre de la figure 2.

Solution : Sa hauteur est 4 et sa taille est 9.

d) Donner les valeurs entières possibles de val pour cet arbre binaire de recherche.

Solution : On a $15 < \text{val} < 18$, donc il reste 16 et 17.

On suppose, pour la suite de cet exercice, que "val" est égal à 16.

2) On rappelle qu'un parcours infixe depuis un nœud consiste, dans l'ordre, à faire un parcours infixe sur le sous-arbre gauche, afficher le nœud puis faire un parcours infixe sur le sous-arbre droit.

Dans le cas d'un parcours suffixe, on fait un parcours suffixe sur le sous-arbre gauche puis un parcours suffixe sur le sous-arbre droit, avant d'afficher le nœud.

a) Donner les valeurs d'affichage des nœuds dans le cas du parcours infixe de l'arbre.

Solution : On obtient 12, 13, 15, 16, 18, 19, 21, 23 et 32.

b) Donner les valeurs d'affichage des nœuds dans le cas du parcours suffixe de l'arbre.

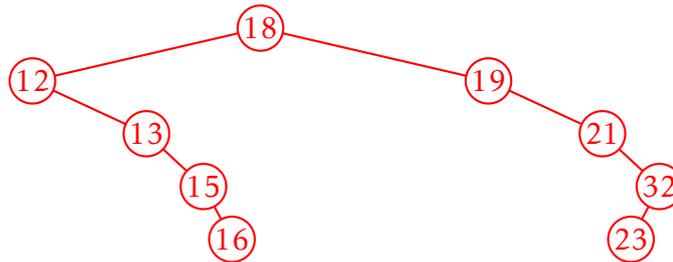
Solution : On obtient 12, 13, 16, 15, 21, 19, 32, 23 et 18.

3) On considère la classe Noeud, définie en Python, donnée à la page suivante.

a) Représenter l'arbre construit suite à l'exécution de l'instruction suivante :

```
racine = Noeud(18)
racine.insere_tout([12, 13, 15, 16, 19, 21, 32, 23])
```

Solution : On obtient :



b) Ecrire les deux instructions permettant de construire l'arbre de la figure 2. On rappelle que le nombre "val" est égal à 16.

Solution : Il faut faire :

```
racine = Noeud(18)
racine.insere_tout([15, 13, 12, 16, 23, 19, 21, 32])
```

c) On considère l'arbre tel qu'il est présenté sur la figure 2. Déterminer l'ordre d'exécution des blocs (repérés de 1 à 3) suite à l'application de la méthode `insere(19)` au nœud racine de cet arbre.

Solution : On passe par les blocs 3, 2 et 1.

4) Ecrire une méthode `recherche(self, v)` qui prend en argument un entier `v` et renvoie la valeur `True` si cet entier est une étiquette de l'arbre, `False` sinon.

Solution :

```

# Version itérative
def recherche(self, v):
    n = self
    while n is not None:
        if v < n.v:
            n = n.ag
        elif v > n.v:
            n = n.ad
        else:
            return True
    return False

# Version réursive
def recherche(self, v):
    if v == self.v:
        return True
    elif v < self.v:
        return self.ag != None and self.ag.recherche(v)
    else:
        return self.ad != None and self.ad.recherche(v)

```

```

class Noeud():
    def __init__(self, v):
        self.ag = None
        self.ad = None
        self.v = v

    def insere(self, v):
        n = self
        est_insere = False
        while not est_insere :
            if v == n.v:
                est_insere = True      ## Bloc 1
            elif v < n.v:
                if n.ag != None:      ##
                    n = n.ag          #
                else:                  # Bloc 2
                    n.ag = Noeud(v)   #
                    est_insere = True ##
            else:
                if n.ad != None:      ##
                    n = n.ad          #
                else:                  # Bloc 3
                    n.ad = Noeud(v)   #
                    est_insere = True ##

    def insere_tout(self, vals):
        for v in vals:
            self.insere(v)

```

EXERCICE 3 : Thèmes abordés : bases de données

Un club de handball souhaite regrouper efficacement toutes ses informations. Il utilise pour cela des bases de données relationnelles afin d'avoir accès aux informations classiques sur les licenciés du club ainsi que sur les matchs du championnat. Le langage SQL a été retenu. On suppose dans l'exercice que tous les joueurs d'une équipe jouent à chaque match de l'équipe.

La structure de la base de données est composée des deux tables (ou relations) suivantes:

Table licenciés		Table matchs	
Attributs	Types	Attributs	Types
id_licencie	INT	id_matchs	INT
prenom	VARCHAR	equipe	VARCHAR
nom	VARCHAR	adversaire	VARCHAR
annee_naissance	INT	lieu	VARCHAR
equipe	VARCHAR	date	DATE

Ci-dessous un exemple de ce que l'on peut trouver dans la base de données :

Exemple **non exhaustif** d'entrées de la table licenciés

id_licencie	prenom	nom	annee_naissance	equipe
63	Jean-Pierre	Masclef	1965	Vétérans
102	Eva	Cujon	1992	Femmes 1
125	Emile	Alinio	2000	Hommes 2
247	Ulysse	Trentain	2008	-12 ans

Exemple **non exhaustif** d'entrées de la table matchs

id_match	equipe	adversaire	lieu	date
746	-16 ans	PHC	Domicile	2021-06-19
780	Vétérans	PHC	Exterieur	2021-06-26
936	Hommes 3	LSC	Exterieur	2021-06-20
1032	-19 ans	LOH	Exterieur	2021-05-22
1485	Femmes 2	CHM	Domicile	2021-05-02
1512	Vétérans	ATC	Domicile	2021-04-12

1) a) L'attribut nom de la table licenciés pourrait-il servir de clé primaire? **Justifier.**

Solution : Il peut y avoir plusieurs joueurs avec le même nom, donc cela ne peut pas être une clé primaire.

b) Citer un autre attribut de cette table qui pourrait servir de clé primaire.

Solution : L'attribut id_licencie peut servir de clé primaire puisqu'il peut être différent pour chaque joueur.

2) a) Expliquer ce que renvoie la requête SQL suivante :

```
SELECT prenom, nom FROM licenciés WHERE equipe = "-12 ans"
```

Solution : On va obtenir le prénom et le nom de tous les joueurs de l'équipe "-12 ans".

b) Que renvoie la requête précédente si prenom, nom est remplacé par une étoile (*)?

Solution : On obtient alors tous les attributs de ces joueurs.

c) Écrire la requête qui permet l'affichage des dates de tous les matchs joués à domicile de l'équipe *Vétérans*.

Solution : On peut faire :

```
SELECT date FROM matchs WHERE equipe = "Vétérans" and lieu = "Domicile"
```

- 3) Écrire la requête qui permet d'inscrire dans la table *licenciers*, *Jean Lavenu* né en 2001 de l'équipe *Hommes 2* et qui aura comme numéro de licence 287 dans ce club.

Solution : On peut faire :

```
INSERT INTO licenciers VALUES (287, "Jean", "Lavenu", 2001, "Hommes 2")
```

- 4) On souhaite mettre à jour les données de la table *licenciers* du joueur *Joseph Cuviller*, déjà inscrit. Il était en équipe *Hommes 2* et il est maintenant en équipe *Vétérans*. Afin de modifier la table dans ce sens, proposer la requête adéquate.

Solution : On peut faire :

```
UPDATE licenciers SET equipe = "Vétérans"  
WHERE prenom = "Joseph" AND nom = "Cuviller"
```

- 5) Pour obtenir le nom de tous les licenciés qui jouent contre le LSC le 19 juin 2021, compléter la requête ci-dessous.

```
SELECT nom FROM licenciers  
JOIN Matches ON licenciers.equipe = matches.equipe  
WHERE adversaire = "LSC" AND date = 2021-06-19
```

Rappels sur SQL

• Types de données

VARCHAR(t)	Texte de t caractères variables.
INT	Nombre entier de -2^{31} à $2^{31} - 1$ (signé) ou de 0 à $2^{32} - 1$ (non signé)
DATE	Date format AAAA-MM-JJ

• Quelques exemples de syntaxe SQL :

```
-- Insérer des enregistrements :  
INSERT INTO Table (attribut1, attribut2) VALUES(valeur1, valeur2);  
-- Modifier des enregistrements :  
UPDATE Table SET attribut1=valeur1, attribut2=valeur2 WHERE Selecteur;  
DELETE FROM Table WHERE Selecteur; -- Supprimer des enregistrements  
SELECT attributs FROM Table WHERE Selecteur; -- Sélectionner des enregistrements  
SELECT attributs FROM TableA -- Effectuer  
JOIN TableB ON TableA.cle1=TableB.cle2 -- une  
WHERE Selecteur; -- jointure
```

EXERCICE 4 : Cet exercice porte sur la programmation en général et la récursivité en particulier.

On considère un tableau de nombres de n lignes et p colonnes.

Les lignes sont numérotées de 0 à $n - 1$ et les colonnes sont numérotées de 0 à $p - 1$. La case en haut à gauche est repérée par $(0,0)$ et la case en bas à droite par $(n - 1, p - 1)$.

On appelle chemin une succession de cases allant de la case $(0,0)$ à la case $(n - 1, p - 1)$, en n'autorisant que des déplacements case par case : soit vers la droite, soit vers le bas.

On appelle somme d'un chemin la somme des entiers situés sur ce chemin.

Par exemple, pour le tableau T suivant :

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

- Un chemin est $(0,0), (0,1), (0,2), (1,2), (2,2), (2,3)$ (en gras sur le tableau).
- La somme du chemin précédent est 14.
- $(0,0), (0,2), (2,2), (2,3)$ n'est pas un chemin.

L'objectif de cet exercice est de déterminer la somme maximale pour tous les chemins possibles allant de la case $(0,0)$ à la case $(n - 1, p - 1)$.

1) On considère tous les chemins allant de la case $(0,0)$ à la case $(2,3)$ du tableau T donné en exemple.

a) Un tel chemin comprend nécessairement 3 déplacements vers la droite. Combien de déplacements vers le bas comprend-il?

Solution : Il faut 2 déplacements vers le bas.

b) La longueur d'un chemin est égal au nombre de cases de ce chemin. Justifier que tous les chemins allant de $(0,0)$ à $(2,3)$ ont une longueur égale à 6.

Solution : Il faut 2 déplacements vers le bas et 3 vers la droite, ce qui fait 5 cases à visiter. En rajoutant le départ, ça fait un chemin de longueur 6.

2) En listant tous les chemins possibles allant de $(0,0)$ à $(2,3)$ du tableau T, déterminer un chemin qui permet d'obtenir la somme maximale et la valeur de cette somme.

Solution : Voici les différentes valeurs possibles :

$$4 + 1 + 1 + 3 + 1 + 1 = 11$$

$$4 + 1 + 1 + 2 + 1 + 1 = 10$$

$$4 + 1 + 1 + 2 + 5 + 1 = 14$$

$$4 + 1 + 0 + 2 + 1 + 1 = 9$$

$$4 + 1 + 0 + 2 + 5 + 1 = 13$$

$$4 + 1 + 0 + 1 + 5 + 1 = 12$$

$$4 + 2 + 0 + 2 + 1 + 1 = 10$$

$$4 + 2 + 0 + 2 + 5 + 1 = 14$$

$$4 + 2 + 0 + 1 + 5 + 1 = 13$$

$$4 + 2 + 3 + 1 + 5 + 1 = 16$$

Le chemin de somme maximale est $(0,0), (1,0), (2,0), (2,1), (2,2), (2,3)$ et vaut 16.

3) On veut créer le tableau T' où chaque élément $T'[i][j]$ est la somme maximale pour tous les chemins possibles allant de $(0,0)$ à (i,j) .

a) Compléter le tableau T' associé au tableau T ci-dessous.

$$T = \begin{array}{|c|c|c|c|} \hline 4 & 1 & 1 & 3 \\ \hline 2 & 0 & 2 & 1 \\ \hline 3 & 1 & 5 & 1 \\ \hline \end{array}$$

$$T' = \begin{array}{|c|c|c|c|} \hline 4 & 5 & 6 & 9 \\ \hline 6 & 6 & 8 & 10 \\ \hline 9 & 10 & 15 & 16 \\ \hline \end{array}$$

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

Solution : Pour aller en $(0, j)$, avec $j > 0$, il faut forcément venir de $(0, j - 1)$.

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

4) Justifier que si i et j sont différents de 0, alors :

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

Solution : Pour aller en (i, j) , avec $i > 0$ et $j > 0$, il faut soit venir de $(i - 1, j)$, soit de $(i, j - 1)$.

Les deux valeurs à comparer sont $T[i][j] + T'[i - 1][j]$ et $T[i][j] + T'[i][j - 1]$.

Comme il faut prendre le plus grand des deux, on obtient l'égalité demandée.

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

5) On veut créer la fonction récursive `somme_max` ayant pour paramètres un tableau `T`, un entier i et un entier j . Cette fonction renvoie la somme maximale pour tous les chemins possibles allant de la case $(0, 0)$ à la case (i, j) .

a) Quel est le cas de base, à savoir le cas qui est traité directement sans faire appel à la fonction `somme_max`? Que renvoie-t-on dans ce cas?

Solution : Dans le cas `somme_max(T, 0, 0)`, on renvoie `T[0][0]`.

b) À l'aide de la question précédente, écrire en Python la fonction récursive `somme_max`.

Solution :

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

c) Quel appel de fonction doit-on faire pour résoudre le problème initial?

Solution : Il faut appeler `somme_max(T, 2, 3)`.