



**NSI - BAC BLANC**  
**Terminale (Correction)**  
Janvier 2026

---

**BACCALAURÉAT GÉNÉRAL**

ÉPREUVE D'ENSEIGNEMENT DE SPÉCIALITÉ

SESSION 2026

**NUMÉRIQUE ET SCIENCES  
INFORMATIQUES**

Durée de l'épreuve : 3 heures 30

L'usage de la calculatrice n'est pas autorisé.

Dès que ce sujet vous est remis, assurez-vous qu'il est complet.

Ce sujet comporte 24 pages numérotées de 1/24 à 24/24.

**Le candidat traite les 3 exercices proposés**

*Merci de rendre le sujet avec la copie*

**Exercice 1.****8 points**

Cet exercice porte sur les arbres binaires, les files et la programmation orientée objet.

Cet exercice comporte deux parties indépendantes.

**PARTIE 1**

Une entreprise stocke les identifiants de ses clients dans un arbre binaire de recherche. On rappelle qu'un arbre binaire est composé de nœuds, chacun des nœuds possédant éventuellement un sous-arbre gauche et éventuellement un sous-arbre droit.

La taille d'un arbre est le nombre de nœuds qu'il contient. Sa hauteur est le nombre de nœuds du plus long chemin qui joint le nœud racine à l'une des feuilles. On convient que la hauteur d'un arbre ne contenant qu'un nœud vaut 1 et celle de l'arbre vide vaut 0.

Dans cet arbre binaire de recherche, chaque nœud contient une valeur, ici une chaîne de caractères, qui est, avec l'ordre lexicographique (celui du dictionnaire) :

- strictement supérieure à toutes les valeurs des nœuds du sous-arbre gauche ;
- strictement inférieure à toutes les valeurs des nœuds du sous-arbre droit.

Ainsi les valeurs de cet arbre sont toutes distinctes. On considère l'arbre binaire de recherche suivant :

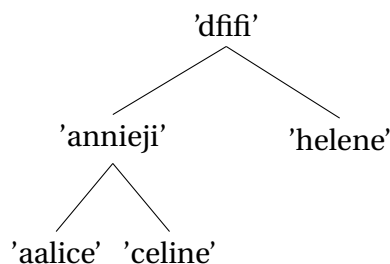


FIGURE 1 – Arbre binaire de recherche.

1. Donner la taille et la hauteur de l'arbre binaire de recherche de la figure 1.

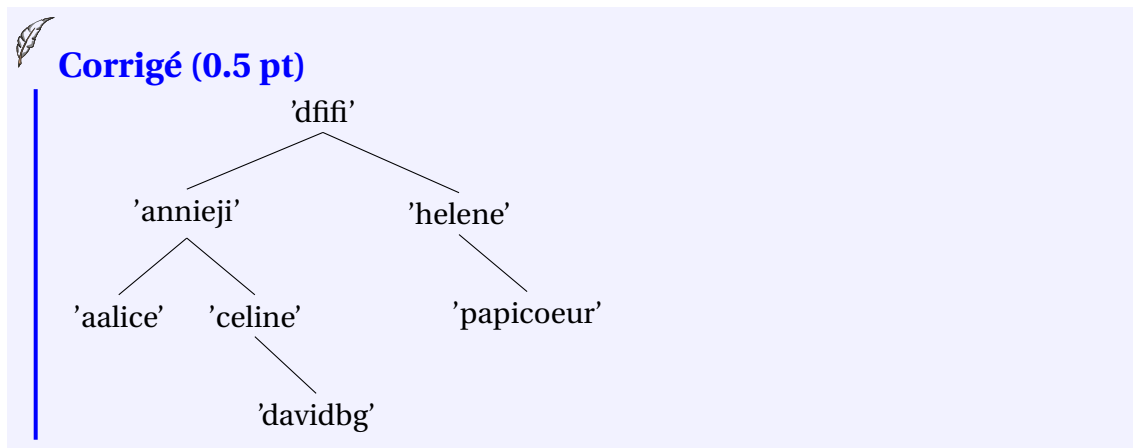
**Corrigé (0.5 pt)**

- Taille : 5 nœuds.
- Hauteur : 3 (par exemple `'dfifi'` → `'annieji'` → `'alice'`).

**Remarque**

Ici la hauteur est définie comme un nombre de nœuds (et non d'arêtes).

2. Recopier cet arbre après l'ajout des identifiants suivants : 'davidbg' et 'papicoeur' dans cet ordre.



3. On décide de parcourir cet arbre pour obtenir la liste des identifiants dans l'ordre lexicographique.

Donnez les listes obtenues pour chacun des 4 parcours A, B, C et D suivants (dans le sens anti-horraire) :

- A - Parcours en largeur d'abord
- B - Parcours en profondeur dans l'ordre préfixe
- C - Parcours en profondeur dans l'ordre infixe
- D - Parcours en profondeur dans l'ordre suffixe (ou postfixe)

Puis en déduire celui (ou ceux) qui permet(tent) d'obtenir l'ordre lexicographique.

**Corrigé (0.75 pt)**

- **A:** 'dfifi', 'annieji', 'helene', 'aalice', 'celine', 'papicoeur', 'davidbg'.
- **B:** 'dfifi', 'annieji', 'aalice', 'celine', 'davidbg', 'helene', 'papicoeur'.
- **C:** 'aalice', 'annieji', 'celine', 'davidbg', 'dfifi', 'helene', 'papicoeur'.
- **D:** 'aalice', 'davidbg', 'celine', 'annieji', 'papicoeur', 'helene', 'dfifi'.

L'ordre lexicographique est obtenu par le parcours infixe : C.



### Remarque

- L'ordre préfixe : on liste chaque sommet la première fois qu'on le rencontre dans la balade.  
(préfixe = 1re rencontre ou à "gauche" dans le sens de parcours (sens anti horaire);)
- L'ordre postfixe (suffixe) : on liste chaque sommet la dernière fois qu'on le rencontre.  
(suffixe = dernière rencontre ou à "droite" (avec fils fantômes) dans le sens de parcours (sens anti horaire);)
- L'ordre infixe : on liste chaque sommet ayant un fils gauche la seconde fois qu'on le voit et chaque sommet sans fils gauche la première fois qu'on le voit.  
(infixe = 2ème rencontre ou en "dessous" (avec fils fantômes) dans le sens de parcours (sens anti horaire))

4. Pour traiter informatiquement les arbres binaires, nous allons utiliser une classe `ABR` .

Un arbre binaire de recherche, nommé `abr` dispose des méthodes suivantes :

- `abr.est_vide()` : renvoie `True` si `abr` est vide et `False` sinon.
- `abr.racine()` : renvoie l'élément situé à la racine de `abr` si `abr` n'est pas vide et un arbre vide sinon.
- `abr.sg()` : renvoie le sous-arbre gauche de `abr` s'il existe et un arbre vide sinon.
- `abr.sd()` : renvoie le sous-arbre droit de `abr` s'il existe et un arbre vide sinon.

On a commencé à écrire une méthode récursive `present` de la classe `ABR` , où le paramètre `identifiant` est une chaîne de caractères et qui retourne `True` si `identifiant` est dans l'arbre et `False` sinon.

Recopier et compléter les lignes 5, 7 et 9 de cette méthode.



### Corrigé (0.75 pt)

Si l'arbre est vide : `False` . Si la racine est `identifiant` : `True` . Si `racine < identifiant` : on cherche dans le sous-arbre droit, sinon dans le sous-arbre gauche.

```

1 def present(self, identifiant):
2     if self.est_vide():
3         return False
4     elif self.racine() == identifiant:
5         return True
6     elif self.racine() < identifiant:
7         return self.sd().present(identifiant)
8     else:
9         return self.sg().present(identifiant)

```

5. Écrire une fonction Python récursive `hauteur_abr` qui prend en paramètre un arbre `abr` (de type `ABR`) et qui renvoie sa hauteur.

**Corrigé (0.75 pt)**

- Si l'arbre est vide, la hauteur vaut 0.
- Sinon, la hauteur vaut  $1 + \max(\text{hauteur}(\text{sg}), \text{hauteur}(\text{sd}))$ .

```
def hauteur_abr(abr: "ABR") -> int:
    """Renvoie la hauteur de l'arbre abr (vide -> 0, un noeud -> 1).
    if abr.est_vide():
        return 0
    return 1 + max(hauteur_abr(abr.sg()), hauteur_abr(abr.sd()))
```

**PARTIE 2**

On considère une structure de données file que l'on représentera par des éléments en ligne, l'élément à droite étant la tête de la file et l'élément à gauche étant la queue de la file.

On appelle `f1` la file suivante :

'bac'	'nsi'	'2023'	'file'	
-------	-------	--------	--------	--

On suppose que les quatre fonctions suivantes ont été programmées préalablement en langage Python :

- `creer_file()` : renvoie une file vide ;
- `est_vide(f)` : renvoie `True` si la file `f` est vide et `False` sinon ;
- `enfiler(f, e)` : ajoute l'élément `e` à la queue de la file `f` ;
- `defiler(f)` : renvoie l'élément situé à la tête de la file `f` et le retire de la file.

Les trois questions suivantes sont indépendantes.

6. Donner le résultat renvoyé après l'appel de la fonction `est_vide(f1)` .

**Corrigé (0.25 pt)**

| `f1` contient des éléments, donc `est_vide(f1)` renvoie `False` .

```
>>> est_vide(f1)
False
```

7. Représenter la file `f1` après l'exécution du code `defiler(f1)` .

**Corrigé (0.5 pt)**

| `defiler` retire l'élément en tête (à droite) : on retire `'file'` . Il reste :

'bac'	'nsi'	'2023'	
-------	-------	--------	--

8. Représenter la file `f2` après l'exécution du code suivant :

```
f2 = creer_file()
liste = ['castor' , 'python' , 'poule']
for elt in liste:
    enfiler(f2 , elt)
```



### Corrigé (0.25 pt)

On enfiler à la queue (à gauche). La tête est à droite, donc :

'poule'	'python'	'castor'
---------	----------	----------

9. Recopier et compléter les lignes 4, 6 et 7 de la fonction `longueur` qui prend en paramètre une file `f` et qui renvoie le nombre d'éléments qu'elle contient. Après un appel à la fonction, la file `f` doit retrouver son état d'origine.



### Corrigé (0.75 pt)

On transfère temporairement les éléments dans une file auxiliaire `g` en comptant, puis on restaure `f` en re-transférant les éléments de `g` vers `f`.

```
1 def longueur(f) :
2     resultat = 0
3     g = creer_file()
4     while not(est_vide(f)) :
5         elt = defiler(f)
6         resultat = resultat + 1
7         enfiler(g, elt)
8     while not(est_vide(g)) :
9         enfiler(f, defiler(g))
10    return resultat
```

10. Un site impose à ses clients des critères sur leur mot de passe. Pour cela il utilise la fonction `est_valide` qui prend en paramètre une chaîne de caractères `mot` et qui retourne `True` si `mot` correspond aux critères et `False` sinon.

```
1 def est_valide(mot) :
2     if len(mot) < 8:
3         return False
4     for c in mot:
5         if c in ['!', '#', '@', ';', ':']:
6             return True
7     return False
```

Parmi les mots de passe suivants, recopier celui qui sera validé par cette fonction.

- A- 'best@'
- B- 'patap23'
- C- '2!@59fgs'

**Corrigé (0.5 pt)**

Le seul mot de passe valide est '2!@59fgs' (longueur 8 et contient '!' et '@').

11. Le tableau suivant montre, sur deux exemples, l'évolution d'une file `f3` après l'exécution de l'instruction `ajouter_mot(f3, 'super')` :

	état initial de <code>f3</code>	état de <code>f3</code> après l'instruction <code>ajouter_mot(f3, 'super')</code>
Exemple 1	<code>'bac' 'nsi' '2023'</code>	<code>'super' 'bac' 'nsi'</code>
Exemple 2	<code>'test' 'info'</code>	<code>'super' 'test' 'info'</code>

Écrire le code de la fonction `ajouter_mot` qui prend en paramètres une file `f` (qui a au plus 3 éléments) et une chaîne de caractères valide `mdp`. Cette fonction met à jour la file `f` en y ajoutant `mdp` et en défilant, si nécessaire, pour conserver au maximum trois éléments.

On pourra utiliser la fonction `longueur`.

**Corrigé (0.75 pt)**

Si la file a moins de 3 éléments, on enfile. Sinon on défile une fois puis on enfile.

```
def ajouter_mot(f, mdp):
    if longueur(f) < 3:
        enfiler(f, mdp)
    else:
        defiler(f)
        enfiler(f, mdp)
```

12. Pour intensifier sa sécurité, le site stocke les trois derniers mots de passe dans une file et interdit au client lorsqu'il change son mot de passe d'utiliser l'un des mots de passe stockés dans cette file.

Recopier et compléter les lignes 7 et 8 de la fonction `mot_file` qui renvoie `True` si `mdp` est présent dans la file `f` et `False` sinon. Après un appel, la file `f` doit retrouver son état d'origine.

**Corrigé (0.5 pt)**

On teste `mdp == elt` et, si c'est vrai, on met `present` à `True`.

```

1 def mot_file(f, mdp):
2     g = creer_file()
3     present = False
4     while not(est_vide(f)):
5         elt = defiler(f)
6         enfiler(g, elt)
7         if mdp == elt:
8             present = True
9     while not(est_vide(g)):
10        enfiler(f, defiler(g))
11    return present

```

13. Écrire une fonction `modification` qui prend en paramètres une file `f` et une chaîne de caractères `nv_mdp`. Si `nv_mdp` est valide et n'appartient pas à la file `f`, alors la fonction met à jour la file et renvoie `True`. Sinon, elle renvoie `False`.

On pourra utiliser les fonctions `mot_file`, `est_valide` et `ajouter_mot`.



### Corrigé (0.75 pt)

On vérifie les deux conditions : `est_valide(nv_mdp)` et `not mot_file(f, nv_mdp)`. Si elles sont vraies, on appelle `ajouter_mot` puis on renvoie `True`. Sinon `False`.

```

def modification(f, nv_mdp):
    if est_valide(nv_mdp) and not mot_file(f, nv_mdp):
        ajouter_mot(f, nv_mdp)
        return True
    return False

```

14. Écrire une fonction Python `file_vers_liste` qui renvoie une liste contenant les éléments de la file `f` dans l'ordre de défilement (de la tête vers la queue) et qui restaure la file `f`.



### Corrigé (0.5 pt)

On défile `f` : on met les éléments dans une liste `res` et dans une file auxiliaire `g`. Puis on restaure `f` en re-défilant `g` vers `f`.

```

from typing import Any, List

def file_vers_liste(f) -> List[Any]:
    """Renvoie la liste des éléments de f (ordre de défilement) et r
    g = creer_file()
    res: List[Any] = []
    while not est_vide(f):
        elt = defiler(f)
        res.append(elt)
        enfiler(g, elt)
    while not est_vide(g):
        enfiler(f, defiler(g))
    return res

```

15. On souhaite éviter toute modification inutile : on veut seulement savoir si deux files  $f$  et  $g$  contiennent exactement les mêmes éléments dans le même ordre.

Écrire une fonction Python `memes_files(f, g)` qui renvoie `True` si c'est le cas et `False` sinon. Après l'appel, les deux files doivent retrouver leur état d'origine.



**Corrigé (0.5 pt)**

```

def memes_files(f, g):
    """Renvoie True si f et g contiennent les mêmes éléments dans le m
    Les deux files sont restaurées après l'appel.
    """
    f1 = file_vers_liste(f)
    f2 = file_vers_liste(g)
    return f1 == f2

```

## Exercice 2.

6 points

Cet exercice porte sur les bases de données et le langage SQL.

On considère une gestion simplifiée des voyages dans l'espace. La base de données utilisée est constituée de quatre relations nommées *Astronaute*, *Fusee*, *Equipe* et *Vol*. Voici le contenu des tables *Astronaute*, *Fusee*, *Equipe* et *Vol*.

Les clés primaires sont soulignées et les clefs étrangères sont précédées d'un # :

Astronaute				
<u>id_astronaute</u>	nom	prenom	nationalite	nb_vols
1	'PESQUET'	'Thomas'	'français'	2
2	'AMSTRONG'	'Neil'	'américain'	8
3	'MAURER'	'Mathias'	'allemand'	1
4	'MCARTHUR'	'Megan'	'américain'	5

Fusee			
<u>id_fusee</u>	modele	constructeur	nb_places
1	'Falcon 9'	'SpaceX'	6
2	'Starship'	'SpaceX'	100
3	'Soyouz'	'TsSKB Progress'	2
4	'SLS'	'Boeing'	6

Equipe	
<u>#id_vol</u>	<u>#id_astronaute</u>
1	1
1	2
1	3
2	1
2	3
3	1
3	2
3	4
4	2
4	4

Vol		
<u>id_vol</u>	<u>#id_fusee</u>	Date
1	1	'12/09/2022'
2	4	'25/10/2022'
3	3	'18/11/2022'
4	2	'23/12/2022'

On pourra utiliser les mots clés suivants :

COUNT, FROM, INSERT, INTO, JOIN, ON, ORDER BY, SELECT, VALUES, WHERE

- Le mot clé **COUNT** permet de récupérer le nombre d'enregistrements issu de la requête. Par exemple, la requête suivante renvoie la valeur 4.

```
SELECT COUNT (*) FROM Astronaute;
```

- Le mot clé **ORDER BY** permet de trier les éléments par ordre alphabétique. Par exemple, la requête suivante :

```
SELECT modele FROM Fusee ORDER BY modele;
```

renvoie la table

'Falcon 9'
'SLS'
'Soyouz'
'Starship'

- Donner les définitions d'une clé primaire et d'une clé étrangère.



### Corrigé (0,5 pt)

Une clé primaire est un attribut (ou un ensemble d'attributs) permettant d'identifier de manière unique chaque enregistrement d'une table. Elle ne peut pas contenir de valeurs identiques ni être vide.

Une clé étrangère est un attribut d'une table qui fait référence à la clé primaire d'une autre table. Elle permet de relier les tables entre elles et d'assurer l'intégrité référentielle.

- Dans la table `Astronaute`, la clé primaire est `id_astronaute`. Expliquer pourquoi la requête suivante renvoie une erreur :

```
INSERT INTO Astronaute
VALUES (3, 'HAIGNERE', 'Claudie', 'français', 3);
```



### Corrigé (0,5 pt)

La requête tente d'insérer un astronaute avec l'identifiant `id_astronaute = 3`. Or cette valeur existe déjà dans la table `Astronaute`. La contrainte d'unicité de la clé primaire est donc violée, ce qui provoque une erreur.

3. Le schéma relationnel de la table `Astronaute` est :

`Astronaute` (`id_astronaute` : INT, `nom` : TEXT, `prenom` : TEXT, `nationalite` : TEXT, `nb_vols` : INT)

Écrire le schéma relationnel de la table `Fusee` en précisant le domaine de chaque attribut.



### Corrigé (0,5 pt)

`Fusee`(`id_fusee` : INT, `modele` : TEXT, `constructeur` : TEXT, `nb_places` : INT)

4. Écrire le résultat que la requête suivante renvoie :

```
SELECT COUNT (*)
FROM Fusee
WHERE constructeur = 'SpaceX';
```



### Corrigé (0,5 pt)

La table `Fusee` contient deux fusées construites par 'SpaceX' ('Falcon 9' et 'Starship'). La requête renvoie donc :

2

5. Écrire une requête SQL qui renvoie le modèle et le constructeur des fusées ayant au moins quatre places.



### Corrigé (0,5 pt)

Il faut sélectionner les attributs `modele` et `constructeur` de la table `Fusee` en ne conservant que les fusées dont `nb_places`  $\geq 4$ .

```
SELECT modele, constructeur
FROM Fusee
WHERE nb_places >= 4;
```

6. Écrire une requête SQL qui renvoie les noms et prénoms des astronautes dans l'ordre alphabétique du nom.



### Corrigé (0,5 pt)

On sélectionne les attributs `nom` et `prenom` de la table `Astronaute` puis on trie les résultats par ordre alphabétique du nom grâce à `ORDER BY`.

```
SELECT nom, prenom
FROM Astronaute
ORDER BY nom ASC;
```

7. Recopier et compléter les requêtes SQL suivantes permettant d'ajouter un cinquième vol avec la fusée 'Soyouz' le '12/04/2023' avec l'équipage composé de PESQUET Thomas et MCARTHUR Megan.



### Corrigé (0.5 pt)

Vol		
<u>id vol</u>	#id_fusee	Date
1	1	'12/09/2022'
2	4	'25/10/2022'
3	3	'18/11/2022'
4	2	'23/12/2022'

On obtient : .

```
INSERT INTO Vol VALUES (5, 3, '12/04/2023');
INSERT INTO Equipe VALUES (5, 1);
INSERT INTO Equipe VALUES (5, 4);
```

8. Écrire une requête SQL permettant d'obtenir le nom et le prénom des astronautes ayant décollé le '25/10/2022' .



### Corrigé (0.5 pt)

Il faut relier les tables `Astronaute` , `Equipe` et `Vol` grâce aux clés étrangères, puis sélectionner les astronautes du vol dont la date vaut '25/10/2022' .

```
SELECT Astronaute.nom, Astronaute.prenom
FROM Astronaute
JOIN Equipe ON Astronaute.id_astronaute = Equipe.id_astronaute
JOIN Vol ON Equipe.id_vol = Vol.id_vol
WHERE Vol.Date = '25/10/2022';
```

9. Écrire une requête SQL qui renvoie, pour chaque vol, la date du vol et le modèle de la fusée utilisée.



### Corrigé (0,5 pt)

La date est stockée dans la table `Vol` et le modèle dans la table `Fusee` . Il faut donc relier ces deux tables puis sélectionner les attributs demandés.

```
SELECT Vol.Date, Fusee.modele
FROM Vol
JOIN Fusee ON Vol.id_fusee = Fusee.id_fusee;
```

10. Écrire une requête SQL qui renvoie le nombre de vols effectués avec une fusée construite par 'SpaceX' .



### Corrigé (0,5 pt)

Il faut relier les tables `Vol` et `Fusee` puis compter les vols associés aux fusées dont le constructeur est 'SpaceX' .

```
SELECT COUNT (*)
FROM Vol
JOIN Fusee ON Vol.id_fusee = Fusee.id_fusee
WHERE Fusee.constructeur = 'SpaceX';
```

11. Écrire une requête SQL qui renvoie le nom et le prénom des astronautes ayant participé à un vol réalisé avec la fusée 'Starship' .

**Corrigé (0,5 pt)**

Il faut relier successivement les tables `Astronaute`, `Equipe`, `Vol` et `Fusee`, puis filtrer sur le modèle `'Starship'`.

```
SELECT Astronaute.nom, Astronaute.prenom
FROM Astronaute
JOIN Equipe ON Astronaute.id_astronaute = Equipe.id_astronaute
JOIN Vol ON Equipe.id_vol = Vol.id_vol
JOIN Fusee ON Vol.id_fusee = Fusee.id_fusee
WHERE Fusee.modele = 'Starship';
```

**12. Algorithmique**

On dispose d'une liste `equipe` issue de la table `Equipe`. Chaque élément est un couple `(id_vol, id_astronaute)`.

**Exemple :** pour la table `Equipe` de l'exercice, on peut représenter la liste par :

$$equipe = [(1, 1), (1, 2), (1, 3), (2, 1), (2, 3), (3, 1), (3, 2), (3, 4), (4, 2), (4, 4)]$$

- L'astronaute d'identifiant `1` apparaît dans les couples :

$$(1, 1), (2, 1), (3, 1)$$

Il a donc participé à 3 vols.

- L'astronaute d'identifiant `4` apparaît dans :

$$(3, 4), (4, 4)$$

Il a donc participé à 2 vols.

Écrire une fonction Python qui prend en paramètres une liste `equipe` et un identifiant `id_astronaute`, et qui renvoie le nombre de vols auxquels cet astronaute a participé.

**Corrigé (0.5 pt)**

On parcourt la liste des couples `(id_vol, id_astronaute)`. Chaque fois que `id_astronaute` correspond à l'identifiant recherché, on incrémente un compteur. À la fin, on renvoie ce compteur.

- Par exemple, avec la liste fournie :

$$nb\_vols\_astronaute(equipe, 1) = \boxed{3} \quad \text{et} \quad nb\_vols\_astronaute(equipe, 4) = \boxed{2}.$$

```
from typing import List, Tuple

def nb_vols_astronaute(equipe: List[Tuple[int, int]], id_astronaute:
    """Renvoie le nombre de vols auxquels l'astronaute id_astronaute

    Paramètres
    -----
    equipe : list[tuple[int, int]]
        Liste de couples (id_vol, id_astronaute).
    id_astronaute : int
        Identifiant de l'astronaute recherché.

    Retour
    -----
    int
        Nombre de vols auxquels l'astronaute a participé.
    """
    compteur: int = 0
    for id_vol, id_ast in equipe:
        if id_ast == id_astronaute:
            compteur += 1
    return compteur
```

**Exercice 3. Protocoles réseaux****6 points**

Cet exercice porte sur l'architecture matérielle, les réseaux, les routeurs et les protocoles de routage.

On considère un réseau local N1 constitué de trois ordinateurs M1, M2, M3 et dont les adresses IP sont les suivantes :

- M1 : 192.168.1.1/24;
- M2 : 192.168.1.2/24;
- M3 : 192.168.2.3/24.

On rappelle que le "/24" situé à la suite de l'adresse IP de M1 signifie que l'adresse réseau du réseau local N1 est 192.168.1.0.

Depuis l'ordinateur M1, un utilisateur exécute la commande ping vers l'ordinateur M3 comme suit :

```
util@M1 ~ % ping 192.168.2.3
PING 192.168.2.3 (192.168.2.3) : 56 data bytes
Hôte inaccessible
```

1. (a) Expliquer le résultat obtenu lors de l'utilisation de la commande ping (on part du principe que la connexion physique entre les machines est fonctionnelle).

**Corrigé (0.5 pt)**

L'adresse IP de M3 n'est pas dans le réseau N1 où se trouve M1. Il ne peut donc pas l'atteindre.

Soit l'adresse de M3 est incorrecte soit M3 n'est pas dans N1.

- (b) Donner le masque de sous-réseau du réseau local N1.

**Corrigé (0.25 pt)**

Le "/24" situé à la suite de l'adresse IP de M1 signifie que le masque de sous-réseau du réseau local N1 est : 255.255.255.0

- (c) Combien d'adresses IP sont disponibles sur le réseaux N1. Expliquez en donnant le rôle des adresses réservées.

**Corrigé (0.75 pt)**

Le "/24" situé à la suite de l'adresse IP de M1 signifie que l'adresse réseau du réseau local N1 est 192.168.1.0 et que seuls les 8 derniers bits sont disponibles. Cependant 2 adresses sont réservées, ce qui fait  $2^8 - 2 = 254$  adresses possibles.

Les deux adresses sont réservées :

- 192.168.1.0 : adresse du réseau;
- 192.168.1.255 : adresse de broadcast;

On ajoute un routeur R1 au réseau N1 :

*"Un routeur moderne se présente comme un boîtier regroupant carte mère, microprocesseur, ROM, RAM ainsi que les ressources réseaux nécessaires (Wi-Fi, Ethernet...). On peut donc le voir comme un ordinateur minimal dédié, dont le système d'exploitation peut être un Linux allégé. De même, tout ordinateur disposant des interfaces adéquates (au minimum deux, souvent Ethernet) peut faire office de routeur s'il est correctement configuré (certaines distributions Linux minimales spécialisent la machine dans cette fonction)."*

Source : Wikipédia, article "Routeur"

2. Définir l'acronyme RAM.



### Corrigé (0.25 pt)

RAM (Random Access Memory)

L'accès à l'information est direct par opposition à un accès séquentiel;

Elle peut être volatile, qui entraîne une perte de toutes les données en mémoire dès qu'elle cesse d'être alimentée en électricité comme les barrettes de PC.

Elle peut être non volatile comme une clef USB.

Deux adresses choisies aléatoirement ont le même temps d'accès d'où le terme de Random

3. Expliquer le terme Linux.



### Corrigé (0.25 pt)

Linux est un système d'exploitation libre

4. Expliquer pourquoi il est nécessaire d'avoir "au minimum deux" interfaces réseau dans un routeur.



### Corrigé

Un routeur est une machine qui connecte au moins deux sous-réseaux. Il faut donc au minimum deux interfaces.

Le réseau N1 est maintenant relié à d'autres réseaux locaux (N2, N3, N4) par l'intermédiaire d'une série de routeurs (R1, R2, R3, R4, R5, R6) :

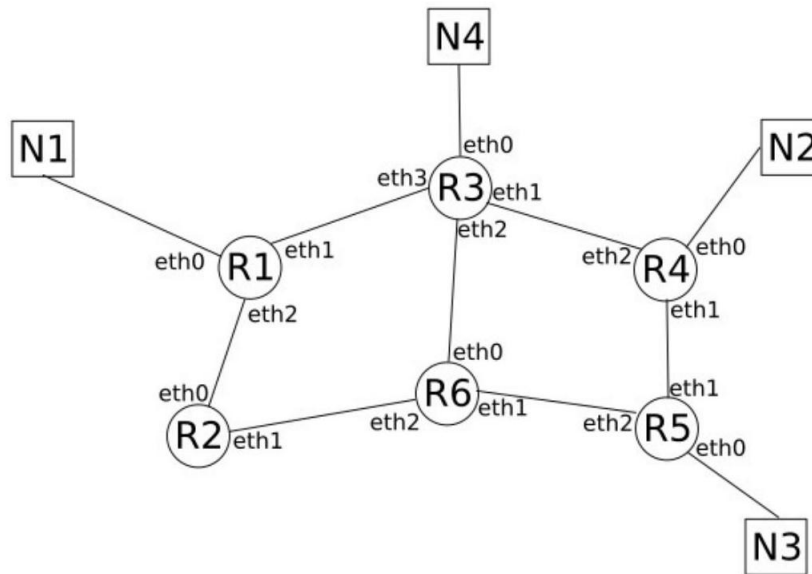


Figure 1. Schéma du réseau

5. Attribuer une adresse IP valide à l'interface eth0 du routeur R1 sachant que l'adresse réseau du réseau N1 est 192.168.1.0.



**Corrigé (0.25 pt)**

| Une adresse IP valide à l'interface eth0 du routeur R1 est 192.168.1.254 .

Dans un premier temps, on utilise le protocole de routage RIP (Routing Information Protocol). On rappelle que dans ce protocole, la métrique de la table de routage correspond au nombre de routeurs à traverser pour atteindre la destination.

La table de routage du routeur R1 est donnée dans le tableau suivant :

Table de routage du routeur R1		
destination	interface de sortie	métrique
N1	eth0	0
N2	eth1	2
N2	eth2	4
N3	eth1	3
N3	eth2	3
N4	eth1	1
N4	eth2	3

6. Déterminer sans justification le chemin parcouru par un paquet de données pour aller d'une machine appartenant au réseau N1 à une machine appartenant au réseau N2.

**Corrigé (0.5 pt)**

De N1 à N2, le chemin parcouru par un paquet de données est R1-R3-R4. C'est le plus court chemin.

Le routeur R3 tombe en panne. Après quelques minutes, la table de routage de R1 est modifiée afin de tenir compte de cette panne.

7. Dresser la table de routage du routeur R1 suite à la panne du routeur R3.

**Corrigé (0.75 pt)**

Table de routage du routeur R1		
destination	interface de sortie	métrique
N1	eth0	0
N2	eth2	4
N3	eth2	3

Le routeur R3 est de nouveau fonctionnel.

Dans la suite de cet exercice, on utilise le protocole de routage OSPF (Open Shortest Path First). On rappelle que dans ce protocole, la métrique de la table de routage correspond à la somme des coûts :  $\text{coût} = \frac{10^8}{d}$  (où  $d$  est la bande passante d'une liaison en bit/s).

Le réseau est constitué de 3 types de liaison de communication :

- Fibre avec un débit de 1 Gbit/s;
- Fast-Ethernet avec un débit de 100 Mbit/s;
- Ethernet avec un débit de 10 Mbit/s.

8. Calculer le coût de chacune de ces liaisons de communication.



### Corrigé (0.75 pt)

- Fibre avec un débit de 1 Gbit/s :  $\text{cout} = \frac{10^8}{10^9} = 0,1$
- Fast-Ethernet avec un débit de 100 Mbit/s :  $\text{cout} = \frac{10^8}{10^8} = 1$
- Ethernet avec un débit de 10 Mbit/s :  $\text{cout} = \frac{10^8}{10^7} = 10$

La table de routage du routeur R1 est donnée dans le tableau suivant :

Table de routage du routeur R1		
destination	interface de sortie	métrique
N1	eth0	0
N2	eth1	10,1
N2	eth2	1,3
N3	eth1	11,1
N3	eth2	0,3
N4	eth1	10
N4	eth2	1,2

D'autre part, le type des différentes liaisons inter-routeurs sont les suivantes :

- R1 - R2 : Fibre;
- R1 - R3 : Ethernet;
- R2 - R6 : INCONNU;
- R3 - R6 : Fast-Ethernet;
- R3 - R4 : Fibre;
- R4 - R5 : Fast-Ethernet;
- R5 - R6 : Fibre.

9. Dédurre de la table de routage de R1 et du schéma du réseau le type de la liaison inter-routeur R2 - R6



### Corrigé (0.5 pt)

Le coût pour rejoindre par exemple le réseau N4 par eth2 est :

$cost_{R1R2} + cost_{R2R6} + cost_{R6R3} = 1,2$  donc  $0,1 + cost_{R2R6} + 1 = 1,2$  donc  $cost_{R2R6} = 0,1$   
donc la liaison R2R6 est de la fibre.

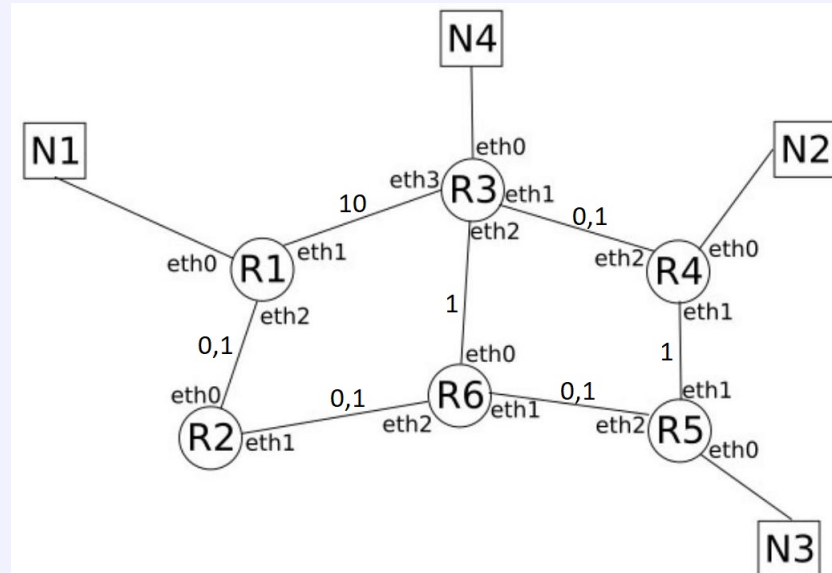


Figure 2. Schéma du réseau avec le coût

Des travaux d'amélioration ont été réalisés sur ce réseau : la liaison inter-routeur R1- R3 est désormais de type Fibre.

10. Modifier la table de routage de R1 en tenant compte de cette amélioration

**Corrigé (0.5 pt)**

Figure 3. Schéma du réseau avec amélioration

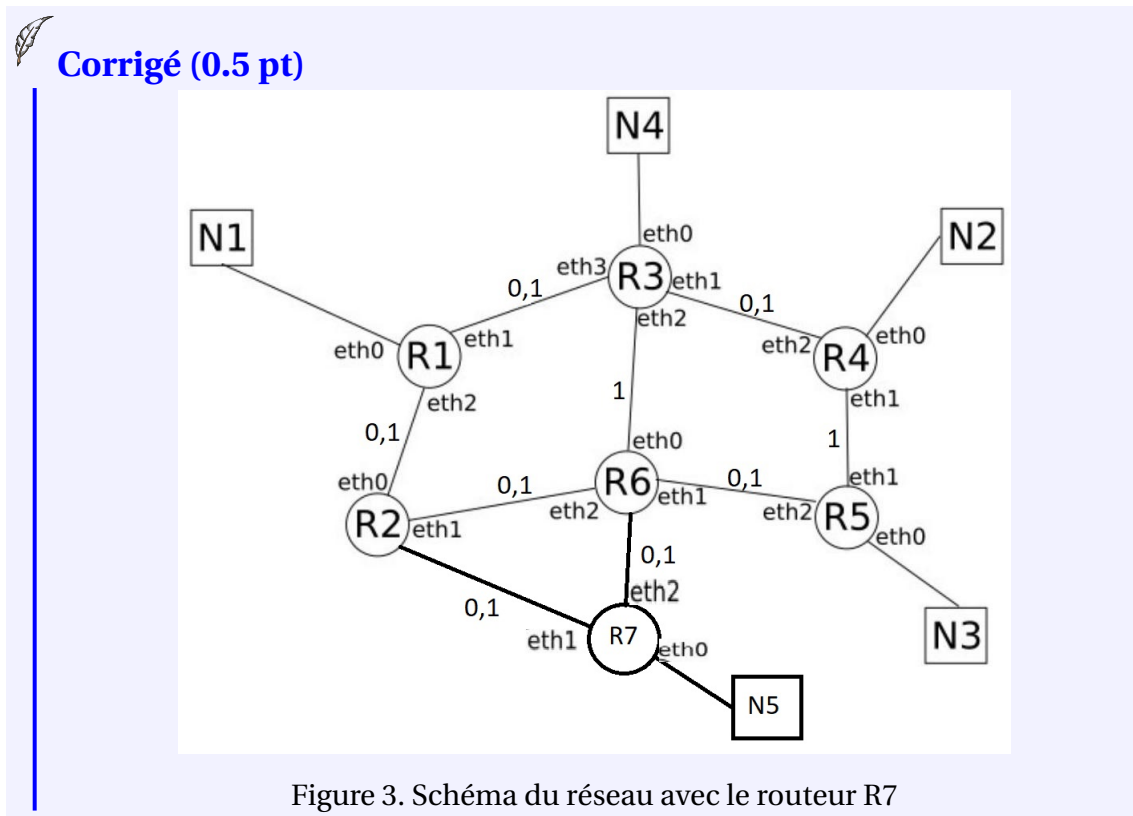
Table de routage du routeur R1		
destination	interface de sortie	métrique
N1	eth0	0
N2	eth1	0,2
N2	eth2	1,3
N3	eth1	1,2
N3	eth2	0,3
N4	eth1	0,1
N4	eth2	1,2

On ajoute un réseau local N5 et un routeur R7 au réseau étudié ci-dessus. Le routeur R7 possède trois interfaces réseaux eth0, eth1 et eth2. eth0 est directement relié au réseau local N5. eth1 et eth2 sont reliés à d'autres routeurs (ces liaisons inter-routeur sont de type Fibre). Les deux tableaux suivants présentent un extrait des tables de routage des routeurs R1 et R3 :

Table de routage du routeur R1		
destination	interface de sortie	métrique
...	...	...
N5	eth1	1,2
N5	eth2	0,2

Table de routage du routeur R3		
destination	interface de sortie	métrieque
...	...	...
N5	eth1	1,3
N5	eth2	1,1
N5	eth3	0,3

11. Recopier et compléter le schéma du réseau (Figure. 1) en ajoutant le routeur R7 et le réseau local N5.



↔ **Fin du devoir** ↔