EXERCICE 1

Cet exercice porte sur l'adressage IP et les protocoles de routage.

Partie A : L'adressage IP

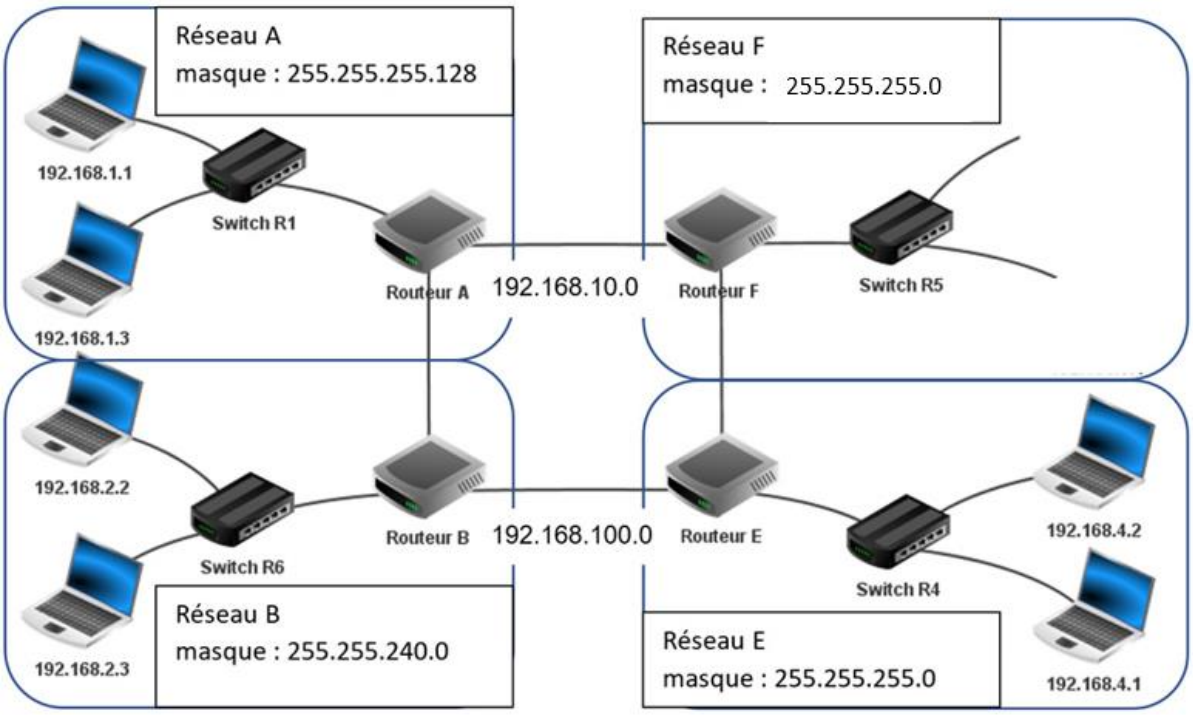

Figure 1. Plan des réseaux étudiés

1. Nous allons considérer le réseau nommé F tel qu'illustré. Son masque de réseau étant, en décimales pointées, 255.255.255.0, les trois premiers octets d'une adresse IP sur ce réseau servent pour la partie réseau de l'adresse (appelée aussi Net ID), le dernier octet sert pour la partie hôte et est propre à chaque machine sur le réseau.
   Une machine connectée au switch R5 possède 192.168.5.3 comme adresse IPV4.
   a. Proposer une adresse IP valide pour le routeur F.
   b. Indiquer le nombre maximum de machines que l'on pourra connecter sur ce réseau F.

2. Pour déterminer la partie d'une adresse IPV4 qui correspond à l'adresse réseau, on effectue un ET logique entre chaque bit de l'adresse IP binaire de l'hôte et celle du masque de sous-réseau. Exemple pour un octet :

	1 1 1 0 1 0 1 0	extrait de l'adresse IP
ET	1 1 1 1 1 0 0 0	extrait du masque du réseau
=	1 1 1 0 1 0 0 0	extrait de l'adresse réseau

On considère le réseau B ;

a. Identifier son masque de sous-réseau sur la figure 1 ci-dessus.

b. Déterminer l'adresse du réseau B, à partir de l'adresse IP d'une machine et du masque de ce réseau. On détaillera soigneusement chaque étape du raisonnement.

c. Proposer un intérêt au fait d'avoir une telle interconnexion entre les quatre routeurs A, B, E et F.

Partie B : Le routage

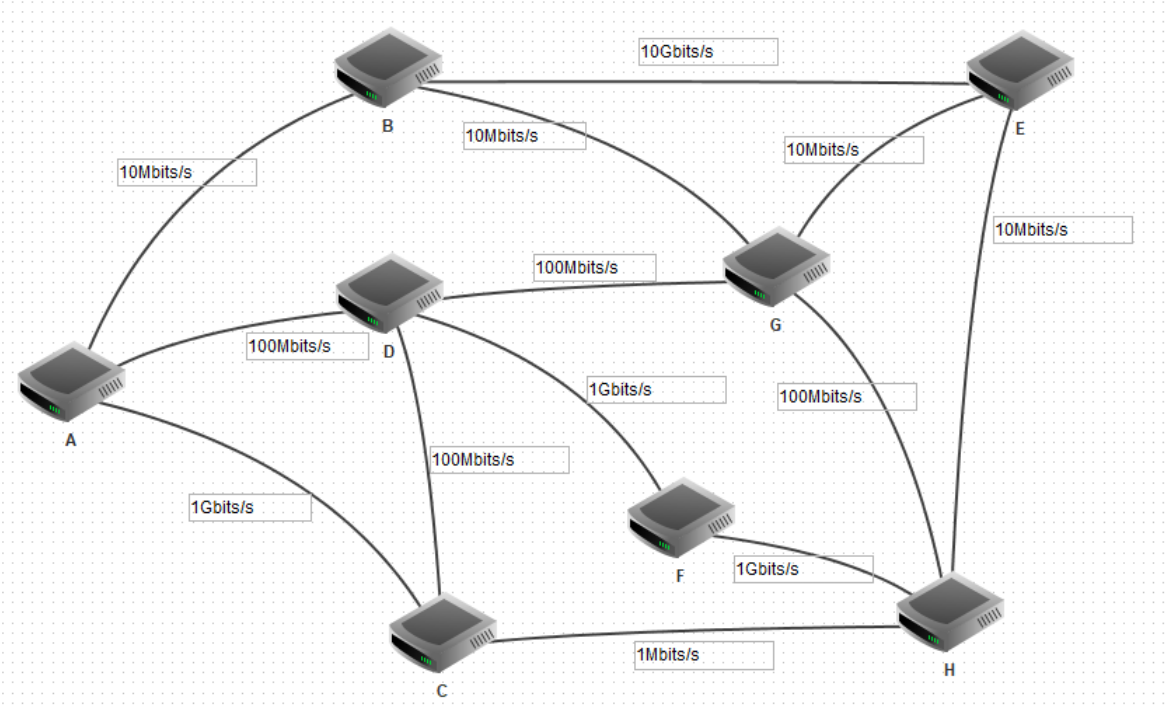

Figure 2. Plan de routage

1. Dans le cadre du protocole RIP, le chemin emprunté par les informations est celui qui aura la distance la plus petite en nombre de sauts. En considérant le réseau présenté ci-dessus :

a. Donner, en respectant le protocole RIP, le(s) chemin(s) possible(s) entre les routeurs A et E, puis entre les routeurs F et B.

b. Recopier sur votre copie les tableaux ci-dessous et les compléter pour le routeur E et le routeur G.

Table de routage du routeur E		
Destination	Routeur suivant	Distance
A	B	2
B		
C		
D		
F		
G		
H		

Table de routage du routeur G		
Destination	Routeur suivant	Distance
A		
B		
C		
D		
E		
F		
H		

2. On considère à présent le protocole OSPF qui se base sur le coût total minimal de la communication. Le coût entre deux routeurs se calcule en fonction du débit selon la formule suivante :

$$coût = \frac{10^8}{débit}$$

a. Recopier et compléter la table de routage du routeur F ci dessous.

Table de routage du routeur F		
Destination	Routeur suivant	Coût total
A	D	1,1
B		10,11
C	D	1,1
D	D	
E	H	10,1
G	D	
H	H	0,1

b. Indiquer quel sera le chemin emprunté par les informations entre le routeur E et le routeur D.

EXERCICE 2

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

L'énoncé de cet exercice utilise les mots-clés du langage SQL suivants : `SELECT`, `FROM`, `WHERE`, `JOIN…ON`, `UPDATE…SET`, `INSERT INTO…VALUES…`, `COUNT`, `ORDER BY`.

La clause `ORDER BY` suivie d'un attribut permet de trier les résultats par ordre croissant de l'attribut.

`SELECT COUNT(*)` renvoie le nombre de lignes d'une requête.

Un zoo souhaite pouvoir suivre ses animaux et ses enclos. Tous les représentants d'une espèce sont réunis dans un même enclos. Plusieurs espèces, si elles peuvent cohabiter ensemble, pourront partager le même enclos.

Il crée une base de données utilisant le langage SQL avec une relation (ou table) **`animal`** qui recense chaque animal du zoo. Vous trouverez un extrait de cette relation ci-dessous (les unités des attributs `age`, `taille` et `poids` sont respectivement ans, m et kg) :

animal					
id_animal	nom	age	taille	poids	nom_espece
145	Romy	18	2.3	130	tigre du Bengale
52	Boris	30	1.10	48	bonobo
…	…	…	…	…	…
225	Hervé	10	2.4	130	lama
404	Moris	6	1.70	100	panda
678	Léon	4	0.30	1	varan

Il crée la relation **`enclos`** dont vous trouverez un extrait ci-dessous (l'unité de l'attribut `surface` est m²) :

enclos				
num_enclos	ecosysteme	surface	struct	date_entretien
40	banquise	50	bassin	04/12/2024
18	forêt tropicale	200	vitré	05/12/2024
…	…	…	…	…
24	savane	300	clôture	04/12/2024
68	désert	2	vivarium	05/12/2024

Il crée également la relation **`espece`** dont vous trouverez un extrait ci-dessous :

espece			
nom_espece	classe	alimentation	num_enclos
impala	mammifères	herbivore	15
ara de Buffon	oiseaux	granivore	77
…	…	…	…
tigre du Bengale	mammifères	carnivore	18
caïman	reptiles	carnivore	45
manchot empereur	oiseaux	carnivore	40
lama	mammifères	herbivore	13

1. Cette question porte sur la lecture et l'écriture de requêtes SQL simples.

   a. Écrire le résultat de la requête ci-dessous.

```
SELECT age, taille, poids FROM animal WHERE nom = 'Moris';
```

   b. Écrire une requête qui permet d'obtenir le nom et l'âge de tous les animaux de l'espèce bonobo, triés du plus jeune au plus vieux.

2. Cette question porte sur le schéma relationnel.

   a. Citer, en justifiant, la clé primaire et la clé étrangère de la relation **`espece`.**

   b. Donner le modèle relationnel de la base de données du zoo. On soulignera les clés primaires et on fera précéder les clés étrangères d'un #.

3. Cette question porte sur les modifications d'une table.

   L'espèce **ornithorynque** a été entrée dans la base comme étant de la **`classe`** des oiseaux alors qu'il s'agit d'un mammifère.

   a. Écrire une requête qui corrige cette erreur dans la table **`espece`.**

   Le couple de lamas du zoo vient de donner naissance au petit lama nommé "Serge" qui mesure 80 cm et pèse 30 kg.

   b. Écrire une requête qui permet d'enregistrer ce nouveau venu au zoo dans la base de données, sachant que les clés primaires de 1 à 178 sont déjà utilisées.

4. Cette question porte sur la jointure entre deux tables

   a. Recopier sur votre feuille la requête SQL et compléter les `...` afin de recenser le nom et l'espèce de tous les animaux carnivores vivant en vivarium dans le zoo.

```
SELECT ...
FROM animal
JOIN espece ON ...
JOIN enclos ON ...
WHERE enclos.struct = 'vivarium' and ...;
```

   On souhaite connaître le nombre d'animaux dans le zoo qui font partie de la classe des **`oiseaux`**.

   b. Écrire la requête qui permet de compter le nombre d'oiseaux dans tout le zoo.

Exercice 3

Cet exercice porte sur les structures de Files

***Simon** est un jeu de société électronique de forme circulaire comportant quatre grosses touches de couleurs différentes : rouge, vert, bleu et jaune.*
Le jeu joue une séquence de couleurs que le joueur doit mémoriser et répéter ensuite. S'il réussit, une couleur parmi les 4 est ajoutée à la fin de la séquence. La nouvelle séquence est jouée depuis le début et le jeu continue. Dès que le joueur se trompe, la séquence est vidée et réinitialisée avec une couleur et une nouvelle partie commence.

Source Wikipédia

Exemple de séquence jouée : rouge→bleu→rouge→jaune→bleu

Dans cet exercice nous essaierons de reproduire ce jeu.
Les quatre couleurs sont stockées dans un tuple nommé `couleurs` :
`couleurs = ("bleu", "rouge", "jaune", "vert")`
Pour stocker la séquence à afficher nous utiliserons une structure de file que l'on nommera `sequence` tout au long de l'exercice.
La file est une structure linéaire de type FIFO (First In First Out). Nous utiliserons durant cet exercice les fonctions suivantes :

Structure de données abstraite : File	
`creer_file_vide()`	: renvoie une file vide
`est_vide(f)`	: renvoie `True` si `f` est vide, `False` sinon
`enfiler(f, element)`	: ajoute `element` en queue de `f`
`defiler(f)`	: retire l'élément en tête de `f` et le renvoie
`taille(f)`	: renvoie le nombre d'éléments de `f`

En fin de chaque séquence, le Simon tire au hasard une couleur parmi les 4 proposées. On utilisera la fonction `randint(a,b)` de la bibliothèque `random` qui permet d'obtenir un nombre entier compris entre **`a`** inclus et **`b`** inclus pour le tirage aléatoire.
Exemple : `randint(1,5)` peut renvoyer 1, 2, 3, 4 ou 5.

1.

Recopier et compléter, sur votre copie, les `...` des lignes 3 et 4 de la fonction `ajout(f)` qui permet de tirer au hasard une couleur et de l'ajouter à une séquence.
La fonction `ajout` prend en paramètre la séquence `f` ; elle renvoie la séquence `f` modifiée (qui intègre la couleur ajoutée au format chaîne de caractères).

```
1  def ajout(f):
2      couleurs = ("bleu", "rouge", "jaune", "vert")
3      indice = randint(...,...)
4      enfiler(...,...)
5      return f
```

En cas d'erreur du joueur durant sa réponse, la partie reprend au début ; il faut donc vider la file `sequence` pour recommencer à zéro en appelant `vider(sequence)` qui permet de rendre la file `sequence` vide sans la renvoyer.

2.

Ecrire la fonction `vider` qui prend en paramètre une séquence `f` et la vide sans la renvoyer.

Le Simon doit afficher successivement les différentes couleurs de la séquence.
Ce rôle est confié à la fonction `affich_seq(sequence)`, qui prend en paramètre la file de couleurs `sequence`, définie par l'algorithme suivant :

- on ajoute une nouvelle couleur à `sequence` ;
- on affiche les couleurs de la séquence, une par une, avec une pause de 0,5 s entre chaque affichage.

Une fonction `affichage(couleur)` (dont la rédaction n'est pas demandée dans cet exercice) permettra l'affichage de la couleur souhaitée avec `couleur` de type chaîne de caractères correspondant à une des 4 couleurs.
La temporisation de 0,5 s sera effectuée avec la commande `time.sleep(0.5)`.
Après l'exécution de la fonction `affich_seq`, la file `sequence` ne devra pas être vidée de ses éléments.

3.

Recopier et compléter, sur la copie, les `...` des lignes 4 à 10 de la fonction `affich_seq(sequence)` ci-dessous :

```
1  def affich_seq(sequence):
2      stock = creer_file_vide()
3      ajout(sequence)
4      while not est_vide(sequence) :
5          c = ...
6          ...
7          time.sleep(0.5)
8          ...
9      while ... :
10         ...
```

4.

Cette question est indépendante des précédentes : bien qu'elle fasse appel aux fonctions construites précédemment, elle peut être résolue même si le candidat n'a pas réussi toutes les questions précédentes.

Nous allons ici créer une fonction `tour_de_jeu(sequence)` qui gère le déroulement d'un tour quelconque de jeu côté joueur. La fonction `tour_de_jeu` prend en paramètre la file de couleurs `sequence`, qui contient un certain nombre de couleurs.

- Le jeu électronique Simon commence par ajouter une couleur à la séquence et affiche l'intégralité de la séquence.
- Le joueur doit reproduire la séquence dans le même ordre. Il choisit une couleur via la fonction `saisie_joueur()`.
- On vérifie si cette couleur est conforme à celle de la séquence.
- S'il s'agit de la bonne couleur, on poursuit sinon on vide `sequence`.
- Si le joueur arrive au bout de la séquence, il valide le tour de jeu et le jeu se poursuit avec un nouveau tour de jeu, sinon le joueur a perdu et le jeu s'arrête.

La fonction `tour_de_jeu` s'arrête donc si le joueur a trouvé toutes les bonnes couleurs de `sequence` dans l'ordre, ou bien dès que le joueur se trompe.

Après l'exécution de la fonction `tour_de_jeu`, la file `sequence` ne devra pas être vidée de ses éléments en cas de victoire.

a. Afin d'obtenir la fonction `tour_de_jeu(sequence)` correspondant au comportement décrit ci-dessus, recopier le script ci-dessous et :

- Compléter le `...`
- Choisir parmi les propositions de syntaxes suivantes lesquelles correspondent aux ZONES A, B, C, D, E et F figurant dans le script et les y remplacer (il ne faut donc en choisir que six parmi les onze) :

```
vider(sequence)
defiler(sequence)
enfiler(sequence,c_joueur)
enfiler(stock,c_seq)
enfiler(sequence, defiler(stock))
enfiler(stock, defiler(sequence))
affich_seq(sequence)
while not est_vide(sequence):
while not est_vide(stock):
if not est_vide(sequence):
if not est_vide(stock):
```

```
1   def tour_de_jeu(sequence):
2       ZONE A
3       stock = creer_file_vide()
4       while not est_vide(sequence) :
5           c_joueur = saisie_joueur()
6           c_seq = ZONE B
7           if c_joueur ... c_seq:
8               ZONE C
9           else:
10              ZONE D
11      ZONE E
12          ZONE F
```

b. Proposer une modification pour que la fonction se répète si le joueur trouve toutes les couleurs de la séquence (dans ce cas, une nouvelle couleur est ajoutée) ou s'il se trompe (dans ce cas, la séquence est vidée et se voit ajouter une nouvelle couleur). On pourra ajouter des instructions qui ne sont pas proposées dans la question a.

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 ?
   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.
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.
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.

$T =$

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

$T' =$

4	5	6	?
6	?	8	10
9	10	?	16

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

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

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 ?
   b) À l'aide de la question précédente, écrire en Python la fonction récursive `somme_max`.
   c) Quel appel de fonction doit-on faire pour résoudre le problème initial ?

Exercice 5 :

L'exercice porte sur les arbres binaires de recherche et la programmation objet.

Dans un entrepôt de e-commerce, un robot mobile autonome exécute successivement les tâches qu'il reçoit tout au long de la journée.

La mémorisation et la gestion de ces tâches sont assurées par une structure de données.

1. Dans l'hypothèse où les tâches devraient être extraites de cette structure (pour être exécutées) dans le même ordre qu'elles ont été mémorisées, préciser si ce fonctionnement traduit le comportement d'une file ou d'une pile. Justifier.

En réalité, selon l'urgence des tâches à effectuer, on associe à chacune d'elles, lors de la mémorisation, un indice de priorité (nombre entier) distinct : il n'y a pas de valeur en double.

Plus cet indice est faible, plus la tâche doit être traitée prioritairement.

La structure de données retenue est assimilée à un arbre binaire de recherche (ABR) dans lequel chaque nœud correspond à une tâche caractérisée par son indice de priorité.

Rappel : Dans un arbre binaire de recherche, chaque nœud est caractérisé par une valeur (ici l'indice de priorité), telle que chaque nœud du sous-arbre gauche a une valeur strictement inférieure à celle du nœud considéré, et que chaque nœud du sous-arbre droit possède une valeur strictement supérieure à celle-ci.
Cette structure de données présente l'avantage de mettre efficacement en œuvre l'insertion ou la suppression de nœuds, ainsi que la recherche d'une valeur.

Par exemple, le robot a reçu successivement, dans l'ordre, des tâches d'indice de priorité 12, 6, 10, 14, 8 et 13. En partant d'un arbre binaire de recherche vide, l'insertion des différentes priorités dans cet arbre donne la figure 1.

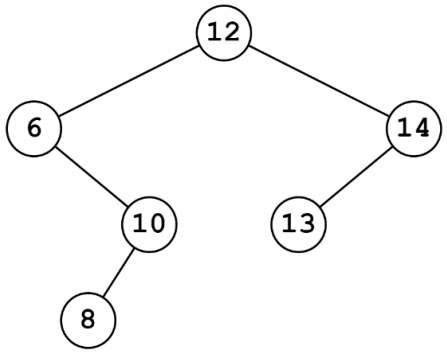

Figure 1 : Exemple d'un arbre binaire

2. En utilisant le vocabulaire couramment utilisé pour les arbres, préciser le terme qui correspond :

- **a.** au nombre de tâches restant à effectuer, c'est-à-dire le nombre total de nœuds de l'arbre ;
- **b.** au nœud représentant la tâche restant à effectuer la plus ancienne ;

c. au nœud représentant la dernière tâche mémorisée (la plus récente).

3. Lorsque le robot reçoit une nouvelle tâche, on déclare un nouvel objet, instance de la classe `Noeud`, puis on l'insère dans l'arbre binaire de recherche (instance de la classe `ABR`) du robot. Ces 2 classes sont définies comme suit :

```
 1  class Noeud:
 2      def __init__(self, tache, indice):
 3          self.tache = tache   #ce que doit accomplir le robot
 4          self.indice = indice #indice de priorité (int)
 5          self.gauche = ABR()  #sous-arbre gauche vide (ABR)
 6          self.droite = ABR()  #sous-arbre droit vide (ABR)
 7
 8
 9  class ABR:
10      #arbre binaire de recherche initialement vide
11      def __init__(self):
12          self.racine = None #arbre vide
13          #Remarque : si l'arbre n'est pas vide, racine est
14          #une instance de la classe Noeud
15
16      def est_vide(self):
17          """renvoie True si l'arbre autoréférencé est vide,
18          False sinon"""
19          return self.racine == None
20
21      def insere(self, nouveau_noeud):
22          """insere un nouveau noeud, instance de la classe
23          Noeud, dans l'ABR"""
24          if self.est_vide():
25              self.racine = nouveau_noeud
26          elif self.racine.indice …… nouveau_noeud.indice
27              self.racine.gauche.insere(nouveau_noeud)
28          else:
29              self.racine.droite.insere(nouveau_noeud)
```

a. Donner les noms des attributs de la classe `Noeud`.

b. Expliquer en quoi la méthode `insere` est dite récursive et justifier rapidement qu'elle se termine.

c. Indiquer le symbole de comparaison manquant dans le test à la **ligne 26** de la méthode `insere` pour que l'arbre binaire de recherche réponde bien à la définition de l'encadré « **Rappel** » de la page 6.

d. On considère le robot dont la liste des tâches est représentée par l'arbre de la figure 1. Ce robot reçoit, successivement et dans l'ordre, des tâches d'indice de priorité 11, 5, 16 et 7, sans avoir accompli la moindre tâche entretemps. Recopier et compléter la figure 1 après l'insertion de ces nouvelles tâches.

4. Avant d'insérer une nouvelle tâche dans l'arbre binaire de recherche, il faut s'assurer que son indice de priorité n'est pas déjà présent.

Écrire une méthode `est_present` de la classe `ABR` qui répond à la description :

```
41  def est_present(self, indice_recherche) :
42      """renvoie True si l'indice de priorité indice_recherche
43      (int) passé en paramètre est déjà l'indice d'un nœud
44      de l'arbre, False sinon"""
```

5. Comme le robot doit toujours traiter la tâche dont l'indice de priorité est le plus petit, on envisage un parcours infixe de l'arbre binaire de recherche.

a. Donner l'ordre des indices de priorité obtenus à l'aide d'un parcours infixe de l'arbre binaire de recherche de la **figure 1**.

b. Expliquer comment exploiter ce parcours pour déterminer la tâche prioritaire.

6. Afin de ne pas parcourir tout l'arbre, il est plus efficace de rechercher la tâche du nœud situé le plus à gauche de l'arbre binaire de recherche : il correspond à la tâche prioritaire.

Recopier et compléter la méthode récursive `tache_prioritaire` de la classe `ABR`:

```
61  def tache_prioritaire(self):
62      """renvoie la tache du noeud situé le plus
63      à gauche de l'ABR supposé non vide"""
64      if self.racine.…….est_vide():#pas de nœud plus à gauche
65          return self.racine.……
66      else:
67          return self.racine.gauche.……()
```

7. Une fois la tâche prioritaire effectuée, il est nécessaire de supprimer le nœud correspondant pour que le robot passe à la tâche suivante :

- Si le nœud correspondant à la tâche prioritaire est une feuille, alors il est simplement supprimé de l'arbre (cette feuille devient un arbre vide)
- Si le nœud correspondant à la tâche prioritaire a un sous-arbre droit non vide, alors ce sous-arbre droit remplace le nœud prioritaire qui est alors écrasé, même s'il s'agit de la racine.

Dessiner alors, pour chaque étape, l'arbre binaire de recherche (seuls les indices de priorités seront représentés) obtenu pour un robot, initialement sans tâche, et qui a, successivement dans l'ordre :

- étape 1 : reçu une tâche d'indice de priorité 14 à accomplir
- étape 2 : reçu une tâche d'indice de priorité 11 à accomplir
- étape 3 : reçu une tâche d'indice de priorité 8 à accomplir
- étape 4 : accompli sa tâche prioritaire
- étape 5 : reçu une tâche d'indice de priorité 12 à accomplir
- étape 6 : accompli sa tâche prioritaire
- étape 7 : accompli sa tâche prioritaire
- étape 8 : reçu une tâche d'indice de priorité 15 à accomplir

- étape 9 : reçu une tâche d'indice de priorité 19 à accomplir
- étape 10 : accompli sa tâche prioritaire