BACCALAURÉAT GÉNÉRAL

ÉPREUVE D'ENSEIGNEMENT DE SPÉCIALITÉ

SESSION 2023

NUMÉRIQUE ET SCIENCES INFORMATIQUES

JOUR 2

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 12 pages numérotées de 1/12 à 12/12.

Le candidat traite les 3 exercices proposés

Exercice 1 (4 points)

Cet exercice traite des bases de données.

Pour gérer son activité, une petite entreprise de travaux d'extérieur à domicile utilise un système de gestion de base de données avec les relations suivantes :

prestations

id_prestation	id_client	date	heure_debut	duree	type	employe
876272	5232	15/06/2020	15 :00	2	tonte	Didier
876273	5231	15/06/2020	15 :30	2	tonte	Sylvie
876274	5229	16/06/2020	09 :00	2	haies	Sylvie
876275	5229	16/06/2020	11 :00	1	tonte	Sylvie
876276	5233	16/06/2020	10 :00	1	tonte	Didier
876277	5228	16/06/2020	14 :00	2	tonte	Didier
876278	5230	16/06/2020	14 :00	2	tonte	Sylvie
876279	5234	17/06/2020	09 :00	3	bucheronnage	Didier
876935	5228	18/11/2020	09 :00	2	haies	Didier
876936	5231	18/11/2020	10 :00	1	nettoyage	Sylvie
876937	5228	18/11/2020	11 :00	1	nettoyage	Didier
876938	5234	18/11/2020	14 :00	4	engazonnement	Sylvie
876939	5233	18/11/2020	14 :00	2	plantations	Didier
876940	5229	19/11/2020	09 :00	2	haies	Sylvie
876941	5230	19/11/2020	09 :00	3	bucheronnage	Didier
876942	5229	19/11/2020	11 :00	1	nettoyage	Sylvie
876943	5235	19/11/2020	14 :00	2	haies	Didier
876944	5232	19/11/2020	14 :00	1	haies	Sylvie
876945	5232	19/11/2020	15 :00	1	plantations	Sylvie

Remarque : la durée est un nombre entier d'heures.

clients

id_client	nom	adresse	code_postal	ville	telephone
5228	Ouellet	8, rue de Verdun	26200	Montélimar	0475016031
5229	Rouze	29, rue Reine Elisabeth	07400	Meysse	0475494977
5230	Bonenfant	58, rue des Soeurs	26780	Espeluche	0475568463
5231	Foucault	67, rue Michel Ange	26200	Montélimar	0475918885
5232	Croteau	98, rue du Gue Jacquet	26200	Montélimar	0475460794
5233	Therriault	11, rue de l'Epeule	26160	Puygiron	0475091013
5234	Rivard	15, rue des Coteaux	26200	Montélimar	0475339127
5235	Blanchard	12, rue du Château	26740	Sauzet	0475305408

1. On n'oubliera pas de justifier les réponses pour les questions a. et b.
   a. Identifier pour chacune de ces deux relations une clé primaire possible.
   b. Identifier la relation qui possède une clé étrangère. Donner cette clé étrangère.
   c. Donner le type pour chacun des deux premiers attributs de la relation `clients`.

On manipule les données en utilisant le langage SQL.

2.
   a. Donner les valeurs renvoyées par la requête suivante :
```
SELECT nom, telephone
FROM clients
WHERE ville='Montélimar';
```

   b. Écrire la requête permettant d'obtenir la date et l'heure de début des prestations de plus d'une heure, réalisées par Didier.

3. Avec les informations présentées en page précédente, donner les valeurs renvoyées par la requête suivante (le mot-clé `DISTINCT` permet d'éviter les doublons dans le résultat de la requête) :

```
SELECT DISTINCT nom
FROM clients JOIN prestations
ON clients.id_client=prestations.id_client
WHERE prestations.employe='Sylvie';
```

4. Pour permettre au logiciel de préparer les factures, on souhaite ajouter le tarif horaire de chaque prestation dans ce jeu de données.
   a. Expliquer quel(s) problème(s) peuvent apparaître si on décide d'ajouter uniquement un attribut `tarif_horaire` à la relation `prestations`.

   b. Proposer un schéma relationnel qui permet d'éviter ce(s) problème(s). On pourra donner la réponse sous la forme :

      Schéma relationnel :

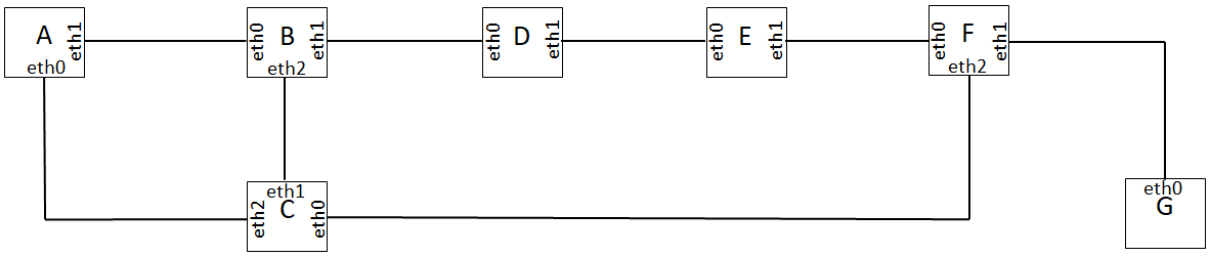
```
tarifs(premier attribut, deuxième attribut,…)
prestations(premier attribut,…)
clients(premier attribut,…)
```

      On indiquera explicitement pour chaque relation les attributs, la clé primaire et les éventuelles clés étrangères (on ne demande pas d'indiquer les types).

Exercice 2 (3 points)

Cet exercice traite des réseaux, de la programmation Python et de l'algorithmique.

1. Pour vendre ses produits, un grossiste en confiseries utilise un service web. Son réseau local correspond au réseau ci-dessous dans lequel les nœuds A, B, C, D, E, F et G sont des routeurs pour lesquels on souhaite déterminer les tables de routage.



Selon le protocole RIP, la distance est le nombre de routeurs traversés par un chemin pour aller d'un routeur à un autre et le chemin choisi entre deux routeurs est celui qui minimise la distance. On exécute ce protocole sur ce réseau.

Reproduire sur la copie et compléter la table suivante, qui indique pour chaque routeur la portion de la table de routage vers le routeur G.

Routeur	destination	passerelle	interface	distance
A	G	C	eth0	2
B	G			
C	G			
D	G			
E	G			
F	G			

Le grossiste propose à ses clients deux assortiments de 100 g chacun :

- le premier « `ToutFruit` » à base de bonbons de différents goûts ;
- le second « `ToutChoc` » à base de chocolats.

Voici un exemple de commande réalisée par un confiseur :

Commande n°343			
Confiserie	**Prix unitaire (€)**	**Quantité**	**Total (€)**
`ToutFruit`	7,00	1	7,00
`ToutChoc`	16,00	3	48,00
	Montant commande (€) :		55,00

2. Pour écouler ses stocks plus aisément, le grossiste propose à ses clients les réductions suivantes :
   - si le montant de la commande est supérieur ou égal à 100 € et strictement inférieur à 200 €, il applique une réduction de 10% ;
   - si le montant de la commande est supérieur ou égal à 200 €, il applique une réduction de 20%.

   Écrire en les complétant les lignes de code, à partir de la ligne 13, de la fonction `calcul_montant` afin de respecter la spécification donnée ci-dessous.

Numéro de lignes	Fonction `calcul_montant`
1	`def calcul_montant(prix_TF, quantite_TF, prix_TC, quantite_TC):`
2	`    """`
3	`    Renvoie le montant de la commande`
4	`    Entrées :`
5	`        prix_TF : prix de ToutFruit`
6	`        quantite_TF : quantite de ToutFruit`
7	`        prix_TC : prix de ToutChoc`
8	`        quantite_TC : quantite de ToutChoc`
9	`    Sortie :`
10	`        montant de la commande apres reduction eventuelle`
11	`    """`
12	`    montant = quantite_TF * prix_TF + quantite_TC * prix_TC`
13	`    if` ***`à compléter`*** `:`
14	`        montant = montant - montant*0.10`
15	`    elif` ***`à compléter`*** `:`
16	`        montant =` ***`à compléter`***
17	`    return` ***`à compléter`***

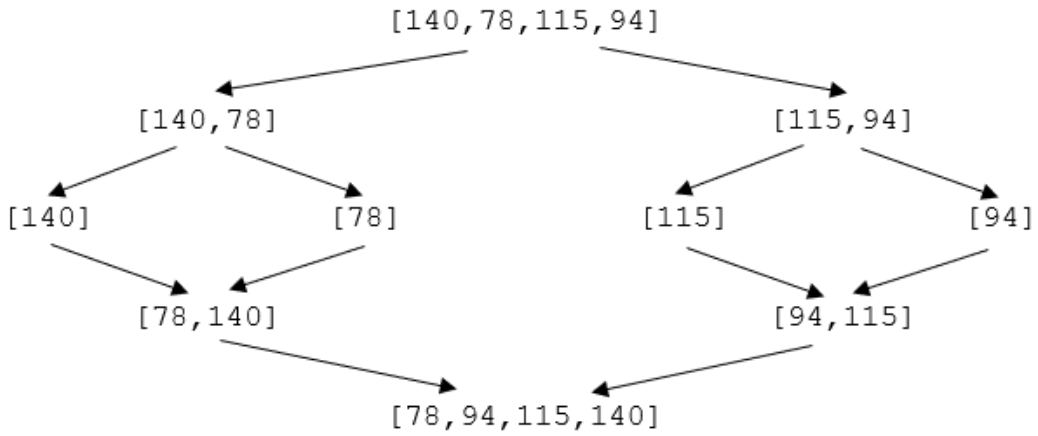
3. Le grossiste enregistre dans la liste `sommes` les montants des commandes, avant réduction, que des clients ont passées dans son magasin.
   Par exemple : `sommes = [140, 78, 115, 94, 46, 108, 55, 53]`

   Il va utiliser la méthode de tri-fusion pour trier la liste `sommes` dans l'ordre croissant.

   L'algorithme du tri-fusion est naturellement décrit de façon récursive (dans toute la suite, on supposera pour simplifier que le nombre d'éléments de la liste est une puissance de 2) :

   - si la liste n'a qu'un ou aucun élément, elle est déjà triée,
   - sinon,
     - on sépare la liste initiale en deux listes de même taille,
     - on applique récursivement l'algorithme sur ces deux listes,
     - on fusionne les deux listes triées obtenues en une seule liste triée.

On trouve ci-dessous un exemple de déroulement de cet algorithme pour la liste `[140,78,115,94]` de 4 éléments :



a. Appliquer sur votre copie, de la même façon que dans l'exemple précédent, le déroulement de l'algorithme pour la liste suivante :

`[46, 108, 55, 53]`

On rappelle que l'appel `L.append(x)` ajoute l'élément `x` à la fin de la liste `L` et que `len(L)` renvoie la taille de la liste `L`.

b. La fonction `fusion` ci-dessous renvoie une liste de valeurs triées à partir des deux listes `liste1` et `liste2` préalablement triées.

Numéro de lignes	Fonction `fusion`
1	`def fusion(liste1, liste2):`
2	`    liste_finale = []`
3	`    i1, i2 = 0, 0`
4	`    à compléter`
5	`        if liste1[i1] <= liste2[i2]:`
6	`            liste_finale.append(liste1[i1])`
7	`            i1 =i1 + 1`
8	`        else:`
9	`            liste_finale.append(liste2[i2])`
10	`            i2 = i2 + 1`
11	`    while i1 < len(liste1):`
12	`        liste_finale.append(liste1[i1])`
13	`        i1 = i1 + 1`
14	`    while i2 < len(liste2):`
15	`        liste_finale.append(liste2[i2])`
16	`        i2 = i2 + 1`
17	`    return liste_finale`

Parmi les quatre propositions suivantes, écrire sur la copie l'instruction à placer à la ligne 4 du code de la fonction `fusion` :

Proposition 1	`while i1 < len(liste1) and i2 < len(liste2):`
Proposition 2	`while i1 < len(liste1) or i2 < len(liste2):`
Proposition 3	`while i1 + i2 < len(liste1) + len(liste2):`
Proposition 4	`while liste1[i1 + 1] < liste2[i2]):`

c. Recopier et compléter sur la copie la fonction récursive `tri_fusion` suivante, qui prend en paramètre une liste non triée et la renvoie sous la forme d'une nouvelle liste triée.

On utilisera la fonction `fusion` de la question précédente.

Exemple : `tri_fusion([140,115,78,94])` doit renvoyer
`[78,94,115,140]`

Aide Python : si `L` est une liste, l'instruction `L[i:j]` renvoie la liste constituée des éléments de `L` indexés de `i` à `j-1`.
Par exemple, si `L=[5,4,8,7,3]`, `L[2:4]` vaut `[8,7]`.

Numéro de lignes	Fonction `tri_fusion`
1	`def tri_fusion(liste):`
2	`    if len(liste) <= 1:`
3	`        return liste`
4	`    à compléter`
...	

Exercice 3 (5 Points)

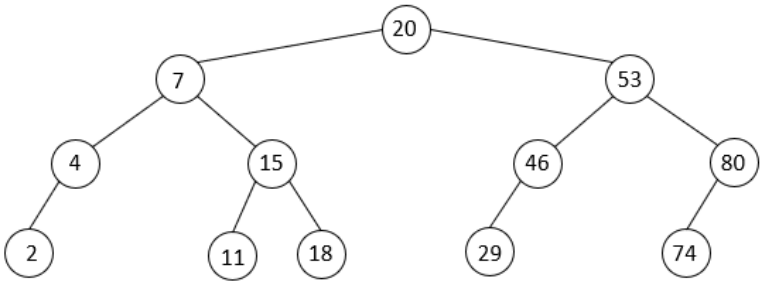
Cet exercice traite des arbres et de l'algorithmique.

Dans cet exercice, la taille d'un arbre est égale au nombre de ses nœuds et on convient que la hauteur d'un arbre ne contenant qu'un nœud vaut 1.

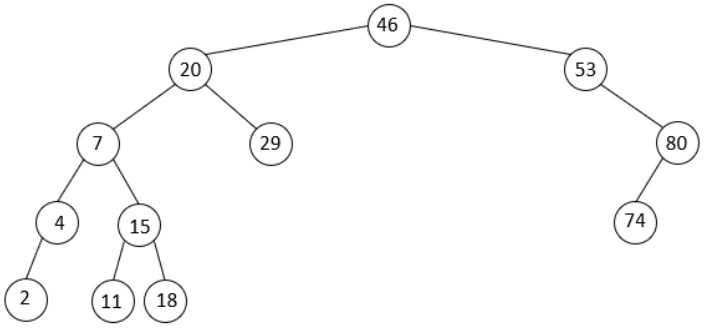
On utilisera la définition suivante : un arbre binaire de recherche est un arbre binaire, dans lequel

- on peut comparer les valeurs des nœuds : ce sont par exemple des nombres entiers, ou des lettres de l'alphabet ;
- si `x` est un nœud de cet arbre et `y` est un nœud du sous-arbre gauche de x, alors il faut que `y.valeur < x.valeur`
- si `x` est un nœud de cet arbre et `y` est un nœud du sous-arbre droit de `x`, alors il faut que `y.valeur` $\geq$ `x.valeur`

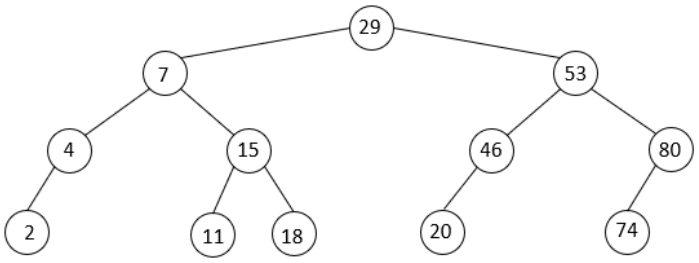
1. Parmi les trois arbres dessinés ci-dessous, recopier sur la copie le numéro correspondant à celui qui n'est pas un arbre binaire de recherche. Justifier.



Arbre 1



Arbre 2



Arbre 3

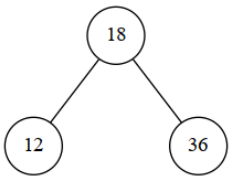
Une classe `ABR`, qui implémente une structure d'arbre binaire de recherche, possède l'interface suivante :

Numéro de lignes	Classe `ABR`
1	`class ABR :`
2	`    def __init__(self, valeur, sa_gauche, sa_droit):`
3	`        self.valeur = valeur #valeur de la racine`
4	`        self.sa_gauche = sa_gauche #sous-arbre gauche`
5	`        self.sa_droit = sa_droit #sous-arbre droit`
6	`    def inserer_noeud(self, valeur):`
7	`        """Renvoie un nouvel ABR avec le nœud de valeur 'valeur'`
8	`        inséré comme nouvelle feuille à sa position correcte"""`
9	`        # code non étudié dans cet exercice`
…	

On prendra la valeur `None` pour représenter un sous-arbre vide.

2. La construction d'un ABR se fait en insérant progressivement les valeurs à partir de la racine : la méthode `inserer_noeud` (dont le code n'est pas étudié dans cet exercice) place ainsi un nœud à sa "bonne place" comme feuille dans la structure, sans modifier le reste de la structure. On admet que la position de cette feuille est unique.

   a. En utilisant les méthodes de la classe ABR :
      - écrire l'instruction Python qui permet d'instancier un objet `a0`, de type ABR, ayant un seul nœud (la racine) de valeur 18.
      - écrire une séquence d'instructions qui permet ensuite d'insérer dans l'objet `a0` les deux feuilles de l'arbre de valeurs 12 et 36.

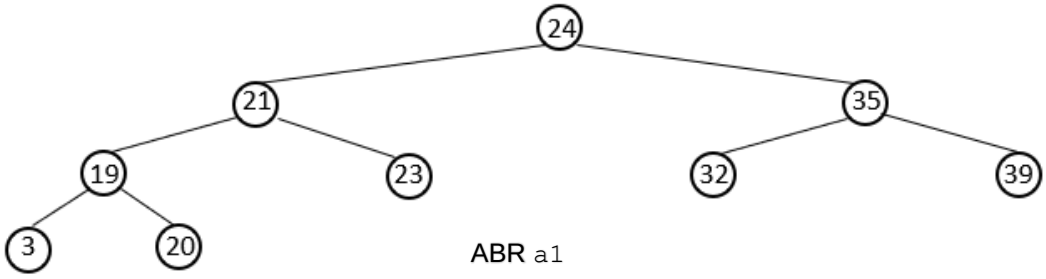


ABR `a0`

   Selon l'ordre dans lequel les valeurs sont insérées, on construit des ABR ayant des structures différentes.

   Voilà par exemple ci-dessous un ABR (nommé `a1`) obtenu en créant une instance de type ABR ayant un seul nœud (la racine) de valeur 24 puis en insérant successivement les valeurs dans l'ordre suivant :

   21 ; 35 ; 19 ; 23 ; 32 ; 39 ; 3 ; 20



ABR `a1`

   b. Dessiner sur la copie l'ABR (nommé `a2`) que l'on obtiendrait en créant une instance de type ABR ayant un seul nœud (la racine) de valeur 3 puis en insérant successivement les valeurs dans l'ordre suivant :

      20 ; 19 ; 21 ; 23 ; 32 ; 24 ; 35 ; 39

   c. Donner la hauteur des ABR `a1` et `a2`.

   d. On complète la classe ABR avec une méthode `calculer_hauteur` qui renvoie la hauteur de l'arbre.
      Recopier sur la copie les lignes 10 et 13 en les complétant par des commentaires et la ligne 14 en la complétant par une instruction dans le code ci-après de cette méthode.
      On pourra utiliser la fonction Python `max` qui prend en paramètres deux nombres et renvoie le maximun de ces deux nombres.

Numéro de lignes	Méthode `calculer_hauteur`
1	`def calculer_hauteur(self):`
2	`    """ Renvoie la hauteur de l'arbre"""`
3	`    if self.sa_droit is None and self.sa_gauche is None:`
4	`    #l'arbre est réduit à une feuille`
5	`        return 1`
6	`    elif self.sa_droit is None`
7	`    #arbre avec une racine et seulement un sous-arbre gauche`
8	`        return 1 + self.sa_gauche.calculer_hauteur()`
9	`    elif self.sa_gauche is None:`
10	`    # à compléter`
11	`         return 1 + self.sa_droit.calculer_hauteur()`
12	`    else:`
13	`    # à compléter`
14	`         return à compléter`

3. La différence de hauteur entre l'ABR `a1` et l'ABR `a2` aura des conséquences lors de la recherche d'une valeur dans l'ABR.

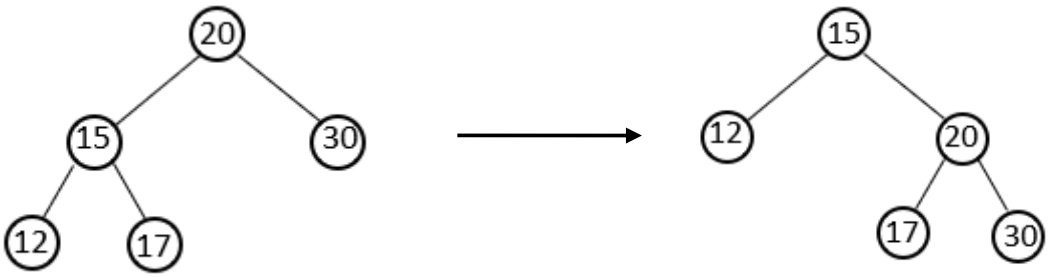
   a. Recopier et compléter sur la copie les lignes 6, 8, 11 et 13 du code ci-dessous de la méthode `rechercher_valeur`, qui permet de tester la présence ou l'absence d'une valeur donnée dans l'ABR :

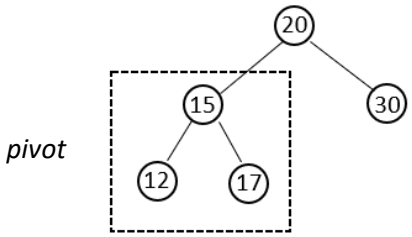
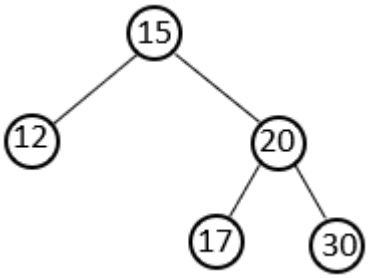
Numéro de lignes	Méthode `rechercher_valeur`
1	`def rechercher_valeur(self, v):`
2	`    """`
3	`    Renvoie True si la valeur v est trouvée dans l'ABR,`
4	`    False sinon`
5	`    """`
6	`    if à compléter`
7	`        return True`
8	`    elif à compléter and self.sa_gauche is not None:`
9	`        return self.sa_gauche.rechercher_valeur(v)`
10	`    elif v > self.valeur and self.sa_droit is not None:`
11	`        return à compléter`
12	`    else:`
13	`        return à compléter`

   b. On admet que le nombre de fois où la méthode `rechercher_valeur` est appelée pour rechercher la valeur 39 dans l'ABR `a2` est 7.
   Donner le nombre de fois où la méthode `rechercher_valeur` est appelée pour rechercher la valeur 20 dans l'ABR `a1`.

4. Il existe des algorithmes pour modifier la structure d'un ABR, afin par exemple de diminuer la hauteur d'un ABR ; on s'intéresse aux algorithmes appelés *rotation*, consistant à faire "pivoter" une partie de l'arbre autour d'un de ses nœuds.

   L'exemple ci-dessous permet d'expliquer l'algorithme pour réaliser une rotation droite d'un ABR autour de sa racine :



On appelle *pivot* le sous-arbre gauche de la racine de l'arbre	
Le sous-arbre droit du pivot devient le sous-arbre gauche de la racine	
La racine ainsi modifiée devient le sous-arbre droit du pivot et la racine du pivot devient la nouvelle racine de l'ABR	

On admet que ces transformations conservent la propriété d'ABR de l'arbre.

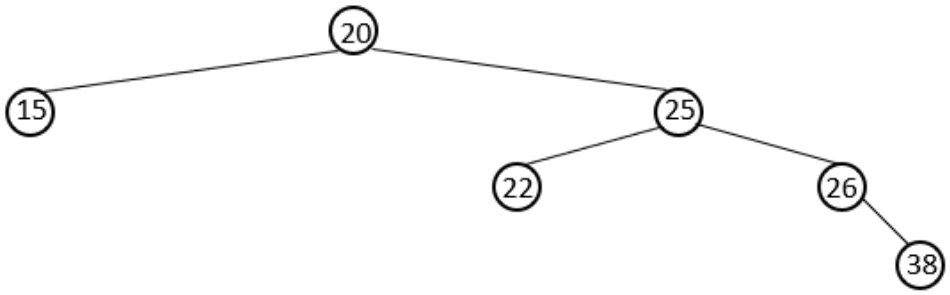
La méthode `rotation_droite` ci-après renvoie une nouvelle instance de type ABR, correspondant à une rotation droite de l'objet de type ABR à partir duquel elle est appelée :

Numéro de lignes	Méthode `rotation_droite`
1	`def rotation_droite(self):`
2	`    """ Renvoie une instance d'un ABR apres une rotation droite`
3	`        On suppose qu'il existe un sous-arbre gauche"""`
4	`    pivot = self.sa_gauche`
5	`    self.sa_gauche = pivot.sa_droit`
6	`    pivot.sa_droit = self`
7	`    return ABR(pivot.valeur,pivot.sa_gauche,pivot.sa_droit)`

Pour réaliser une rotation gauche, on suivra alors l'algorithme suivant :

- on appelle *pivot* le sous-arbre droit de la racine de l'arbre,
- le sous-arbre gauche du pivot devient le sous-arbre droit de la racine,
- la racine ainsi modifiée devient le sous-arbre gauche du pivot et la racine du pivot devient la nouvelle racine de l'ABR

a. En suivant les différentes étapes de cet algorithme, dessiner l'arbre obtenu après une rotation gauche de l'ABR suivant :



b. Écrire le code d'une méthode Python `rotation_gauche` qui réalise la rotation gauche d'un ABR autour de sa racine.