Bac Général Maths + NSI (Mathématiques et Numérique et sciences informatiques)

SUJET D'EXAMEN (SESSION OFFICIELLE)

Numérique et sciences informatiques - Session 2022

Sujet de l'examen national tel que publié pour la session indiquée. Source : sujets nationaux publiés par le ministère chargé de l'examen et les services académiques. Document mis à disposition par FormaV pour la révision.

Toutes les annales et leurs corrigés : https://bac-maths-nsi.fr/annales-sujets-et-corriges

BACCALAURÉAT GÉNÉRAL

ÉPREUVE D'ENSEIGNEMENT DE SPÉCIALITÉ

SESSION 2022

NUMÉRIQUE ET SCIENCES INFORMATIQUES

Durée de l'épreuve : **3 heures 30**

L'usage de la calculatrice et du dictionnaire 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 au choix 3 exercices parmi les 5 exercices proposés.

Chaque exercice est noté sur 4 points.

Exercice 1 (4 points)

Cet exercice traite de programmation, d'algorithmique et de complexité.

On souhaite rechercher dans un tableau les k plus proches voisins d'un objet donné.

On dispose pour cela d'un tableau `t` non vide contenant des objets d'un même type et d'une fonction `distance` qui renvoie la distance entre deux objets quelconques de ce type.
Étant donné un objet `cible` du même type que ceux du tableau `t`, on cherche à déterminer les indices des k éléments du tableau `t` qui sont les plus proches de cet objet (c'est-à-dire ceux dont la distance à l'objet `cible` est la plus petite).

1. On suppose dans cette question que $k = 1$.
   La fonction `plus_proche_voisin(t, cible)` ci-dessous prend en argument le tableau `t` et l'objet `cible`. Écrire sur votre copie uniquement le bloc d'instructions manquant pour que la fonction renvoie l'indice d'un plus proche voisin de `cible`.

```
def plus_proche_voisin(t, cible) :
    dmin = distance(t[0], cible)
    idx_ppv = 0
    n = len(t)
    for idx in range(1, n) :
                                  } bloc à écrire
    return idx_ppv
```

2. On considère que le coût en temps du bloc manquant est constant. Quelle est la complexité de la fonction `plus_proche_voisin` quand $k = 1$?

Dans la suite, on suppose que $k \geq 1$.

3. Une approche naïve consiste à parcourir le tableau `t` pour trouver l'indice de l'élément le plus proche de `cible`, puis à recommencer pour trouver l'indice du deuxième élément le plus proche de `cible`, et ainsi de suite. Cela implique de parcourir k fois tout le tableau.
   Afin de réduire le nombre d'appels à la fonction `distance`, la stratégie suivante permet de ne parcourir le tableau `t` qu'une seule fois. Lors de ce parcours, on stocke dans une liste `kppv`, initialement vide, les tuples `(idx, d)` où `idx` est l'indice d'un k plus proche voisin de `cible` déjà rencontré et `d` la distance correspondante, triés dans l'ordre décroissant de leur distance à `cible`.
   La fonction `recherche_kppv(t, k, cible)` ci-après renvoie ainsi la liste des tuples `(idx, d)` où `idx` est l'indice d'un k plus proche voisin de `cible` dans le tableau `t` et `d` la distance correspondante.

On admet que la fonction `insertion(kppv, idx, d)` insère le tuple `(idx, d)` dans la liste `kppv` de sorte que celle-ci demeure triée dans l'ordre décroissant des distances.

```
def recherche_kppv(t, k, cible):
    kppv = []
    n = len(t)
    for idx in range(n):
        obj = t[idx]
        if len(kppv)<k:
            insertion(kppv, idx, distance(obj, cible))
        else:
            i0, d0 = kppv[0]
            if distance(obj, cible)<d0:
                kppv.pop(0) # supprime le 1er élément de kppv
                insertion(kppv, idx, distance(obj, cible))
    return kppv
```

a. On remarque qu'il y a plusieurs appels identiques à la fonction `distance(obj,cible)`. Comment ne faire qu'un seul appel de cette fonction ?

b. Expliquer l'intérêt de maintenir la liste `kppv` triée.

c. Écrire une fonction `insertion(kppv, idx, d)` qui insère le tuple `(idx, d)` dans la liste `kppv` préalablement triée en préservant l'ordre décroissant selon l'élément `d`.

On pourra éventuellement utiliser la méthode `insert` dont la documentation, fournie par la commande `help(list.insert)`, est la suivante :

```
insert(self, index, object, /)
 Insère l'objet object avant la position index dans l'objet
appelant référencé par self.
```

Exemple d'utilisation de la méthode `insert` :

```
>>> liste = [4, 2, 8, 9]
>>> liste.insert(1, 3)
>>> liste
[4, 3, 2, 8, 9]
```

Exercice 2 (4 points)

Cet exercice porte sur les réseaux et le routage.
Les deux parties de cet exercice sont indépendantes.

Partie A

À son domicile, une élève remarque que l'adresse IP de l'interface réseau (carte wifi) de son ordinateur personnel est 192.168.1.14 avec le masque 255.255.255.0.

Pour chacune des questions ci-dessous, recopier la seule bonne réponse.

1. Sous Unix, quelle instruction en ligne de commande a pu délivrer cette information ?
   - `ifconfig`
   - `ping`
   - `ps`
   - `ls`

2. Parmi les protocoles ci-dessous, quel est celui qui a permis d'attribuer automatiquement cette adresse IP ?
   - DNS
   - DHCP
   - TCP
   - HTTP

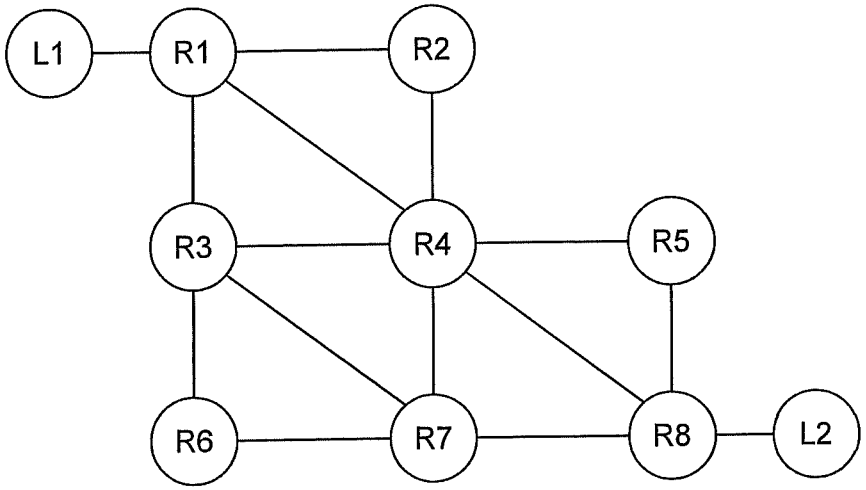
3. Parmi les adresses IP ci-dessous, quelle est la seule possible pour un autre appareil connecté au même réseau ?
   - 192.168.0.14
   - 192.168.0.1
   - 192.168.1.1
   - 192.168.1.255

4. Toujours à son domicile, l'élève consulte une page web qui prétend que l'adresse IP de son ordinateur est 88.168.10.210.
   - C'est une fausse information car son adresse IP est 192.168.1.14 .
   - C'est sûrement faux car seul le fournisseur d'accès peut avoir connaissance de cette information.
   - C'est possible et cette adresse serait celle de la box vers Internet.
   - C'est possible, mais cela signifierait que l'ordinateur est infecté par un malware.

5. Est-il possible qu'un ordinateur connecté au réseau du lycée possède la même adresse IP que l'élève à son domicile ?
   - Oui, à condition que les connexions n'aient pas lieu au même moment.
   - Oui, car les adresses 192.168.x.x ne sont pas routées sur Internet.
   - Oui, à condition d'utiliser un VPN.
   - Non, car deux machines sont identifiées de manière unique par leur adresse IP.

Partie B

On représente ci-dessous un réseau dans lequel R1, R2, R3, R4, R5, R6, R7 et R8 sont des routeurs. Le réseau local L1 est relié au routeur R1 et le réseau local L2 au routeur R8.



Les liaisons sont de trois types :

- Eth : Ethernet, dont la bande passante est de 10 Mb/s ;
- V1 : VDSL, dont la bande passante est de 50 Mb/s ;
- V2 : VDSL2, dont la bande passante est de 100 Mb/s.

On rappelle que la bande passante d'une liaison est la quantité d'information qui peut être transmise en bits/s.
Le tableau ci-dessous précise les types des liaisons entre les routeurs.

Liaison	R1-R2	R1-R3	R1-R4	R2-R4	R3-R4	R3-R6	R3-R7	R4-R5	R4-R7	R4-R8	R5-R8	R6-R7	R7-R8
Type	Eth	V2	Eth	V2	Eth	V2	Eth	V1	V2	Eth	V1	V2	Eth

Pour tenir compte du débit des liaisons, on décide d'utiliser le protocole OSPF (distance liée au coût minimal des liaisons) pour effectuer le routage.
Le coût C d'une liaison est donné par la formule : $C = \frac{10^9}{BP}$ où BP est la bande passante de la liaison en bits/s.

1. Vérifier que le coût d'une liaison VDSL est égal à 20.

2.
   a. Recopier le graphe sur votre copie en inscrivant les coûts des liaisons.

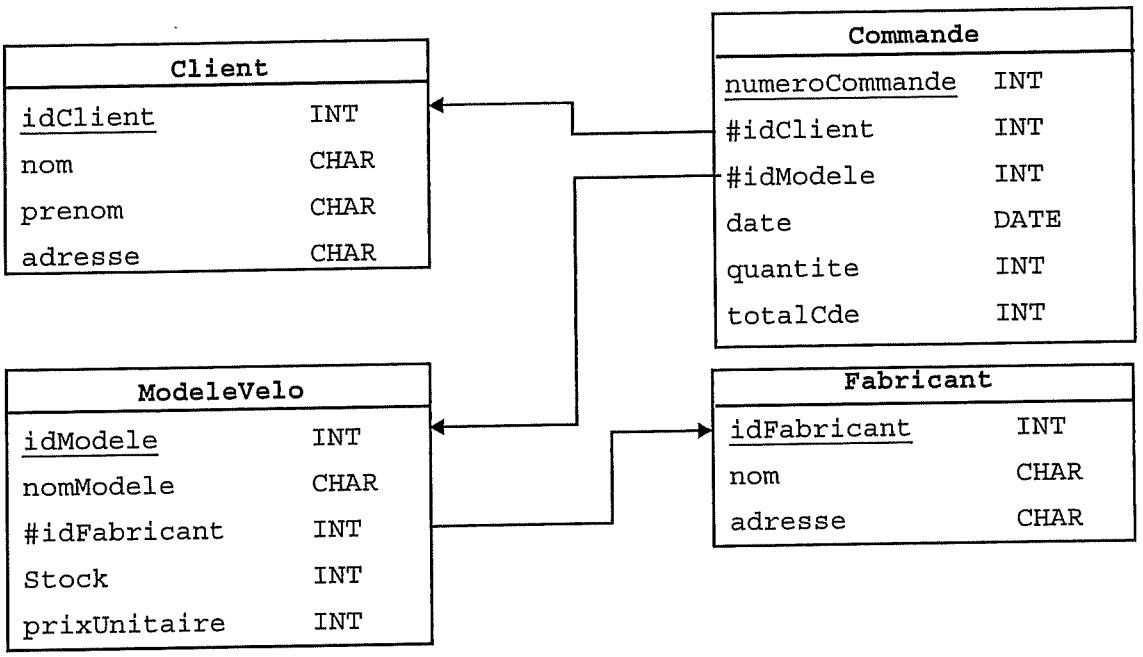
   b. Déterminer le chemin parcouru par un paquet partant du réseau L1 et arrivant au réseau L2, en utilisant le protocole OSPF.

   c. La liaison R1-R4 est remplacée par une liaison de type ADSL avec une bande passante intermédiaire entre celles de type Ethernet et VDSL.
   Quel devrait être le coût maximal de cette liaison pour que des paquets issus du réseau L1 à destination du réseau L2 transitent par celle-ci ? En déduire la bande passante minimale de cette liaison.

Exercice 3 (4 points)

Cet exercice porte sur les bases de données.

Une entreprise vend en ligne des vélos électriques. Afin de gérer le stock des vélos disponibles à la vente, le gestionnaire du site a créé une base de données contenant les quatre relations du schéma relationnel ci-dessous.



Dans le schéma relationnel précédent, un attribut souligné indique qu'il s'agit d'une clé primaire. Un attribut précédé du symbole # indique qu'il s'agit d'une clé étrangère et la flèche associée indique l'attribut référencé. Ainsi, par exemple, l'attribut `idFabricant` de la relation `ModeleVelo` est une clé étrangère qui fait référence à l'attribut `idFabricant` de la relation `Fabricant`.

Dans la suite, les mots clés suivants du langage SQL pourront être utilisés dans les requêtes : `SELECT`, `FROM`, `WHERE`, `JOIN`, `ON`, `DELETE`, `UPDATE`, `SET`, `INSERT`, `AND`, `OR`.

Les fonctions d'agrégation `MIN(att)`, `MAX(att)` et `COUNT(att)` renvoient respectivement la plus petite valeur, la plus grande valeur et le nombre d'enregistrements de l'attribut `att` pour les enregistrements sélectionnés. Ainsi la requête `SELECT MAX(totalCde) FROM Commande` renvoie la plus grande valeur de l'attribut `totalCde` de la table `Commande`.

1. Quand l'entreprise vend le dernier exemplaire en stock d'un modèle, l'attribut `Stock` de la table `ModeleVelo` du modèle correspondant doit être mis à jour pour contenir la valeur 0.
   L'entreprise vient de vendre le dernier exemplaire en stock du modèle `'Bovelo'`.
   Compléter la requête SQL afin de mettre à jour la relation `ModeleVelo`.

```
UPDATE ... SET ... WHERE ...;
```

2. Ravel, un nouveau fabricant, a livré à l'entreprise 10 vélos du modèle `'Bovelo'`. Parmi les cinq requêtes numérotées ci-dessous, en sélectionner deux qui permettent d'intégrer cette nouvelle information et indiquer l'ordre dans lequel elles doivent être saisies.

Requête 1 :
```
UPDATE ModeleVelo SET quantite=10 WHERE nomModele='Bovelo';
```

Requête 2 :
```
INSERT INTO ModeleVelo VALUES (6298, 'Bovelo', 3127, 10,
1990);
```

Requête 3 :
```
UPDATE ModeleVelo SET quantite=10
    WHERE nomModele='Bovelo' AND nomFabricant='Ravel';
```

Requête 4 :
```
INSERT INTO Fabricant
    VALUES (3127, 'Ravel', '89 cours de Vincennes,  75020
Paris');
```

Requête 5 :
```
UPDATE ModeleVelo SET quantite=10 WHERE idFabricant=3127;
```

3. Écrire des requêtes SQL permettant d'obtenir les informations suivantes :

   a. les noms des modèles en rupture de stock et l'identifiant de leur fabricant respectif ;

   b. le nombre de commandes passées depuis le `'2022-01-01'` inclus.
   On précise que le type `DATE` permet de représenter une date par une chaîne de caractère au format `'AAAA-MM-JJ'` et que ces dates peuvent être comparées à l'aide des opérateurs usuels (`=`, `!=`, `>`, `<`, `<=`, `>=`) ;

   c. les noms des fabricants dont le stock de vélos est strictement positif.

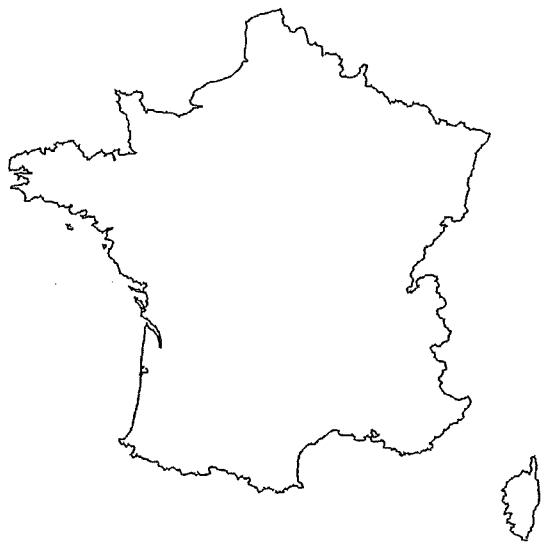
4. Que permet d'obtenir la requête SQL suivante ?

```
SELECT DISTINCT Client.nom
FROM Client
JOIN Commande ON Client.idClient = Commande.idClient
JOIN ModeleVelo ON ModeleVelo.idModele = Commande.idModele
WHERE ModeleVelo.nomModele = 'Bovelo';
```

Exercice 4 (4 points)

Cet exercice porte sur la programmation en Python, la récursivité et la méthode "diviser pour régner".

Une ligne polygonale est constituée d'une liste ordonnée de points, appelés sommets, joints par des segments. L'algorithme de Douglas-Peuker permet de simplifier une ligne polygonale en supprimant certains de ses sommets. L'effet de l'algorithme appliqué aux lignes polygonales du contour de la France métropolitaine est illustré ci-dessous.



Avant application de l'algorithme



Après application de l'algorithme

On implémentera cet algorithme dans la dernière question de l'exercice. Pour cela nous allons d'abord implémenter des fonctions auxiliaires.
On suppose dans la suite que les sommets sont des points du plan dont les coordonnées (x, y) dans un repère orthonormé fixé sont représentées par des tuples de longueur 2.

1. La distance qui sépare deux points A et B de coordonnées (x_A, y_A) et (x_B, y_B) est donnée par la formule $\sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$.
   On rappelle que la fonction `sqrt` du module `math` de Python renvoie la racine carrée d'un nombre positif ou nul.
   a. Écrire une instruction qui permet d'importer la fonction `sqrt` du module `math`.
   b. Supposant l'import réalisé, écrire une fonction `distance_points(a,b)` qui prend en argument deux tuples `a` et `b` représentant les coordonnées de deux points et renvoie la distance qui les sépare.

2. On dispose d'une fonction `distance_point_droite(p, a, b)` qui prend en argument les tuples représentant les coordonnées respectives des points P, A et B, et qui renvoie la distance du point P à la droite (AB). L'exécution de cette fonction produit une erreur dans le cas où les points A et B sont égaux.
   À l'aide des fonctions `distance_points` et `distance_point_droite`, écrire une fonction `distance(p,a,b)` qui renvoie la distance entre le point P et la droite (AB) si les points A et B sont distincts et la distance AP sinon.

Dans la suite, on dira que la fonction `distance` calcule la distance entre le point P et les points A et B, éventuellement confondus.

3. On a besoin d'une fonction `le_plus_loin(ligne)` qui prend en argument une liste de tuples représentant les coordonnées des points d'une ligne polygonale.
Cette fonction doit renvoyer un tuple composé de :
- l'indice du point de coordonnées `p` de la ligne polygonale d'extrémités `deb` et `fin`, pour lequel la distance `distance(p, deb, fin)` est la plus grande ;
- la valeur correspondante de cette distance.
On fournit le code incomplet suivant :

```
def le_plus_loin(ligne):
    n = len(ligne)
    deb = ligne[0]
    fin = ligne[n-1]
    dmax = 0
    indice_max = 0
    for idx in range(1, n-1):
        p = ...
        d = distance(p, deb, fin)
        if ...:
            ...
            ...
    return ...
```

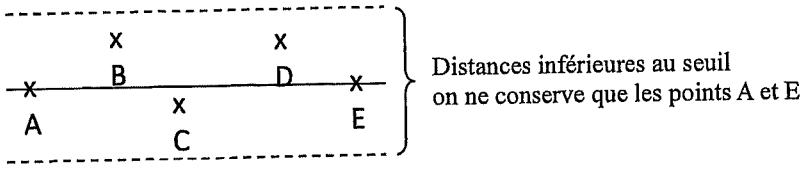
Recopier et compléter le code de cette fonction.

4. Écrire une fonction `extrait(tab, i, j)` qui renvoie la copie du tableau `tab` des cases d'indice `i` inclus à `j` inclus pour `0 ≤ i ≤ j < len(tab)`.
L'appel `extrait([7, 4, 9, 12], 2, 3)` renverra ainsi `[9, 12]`.

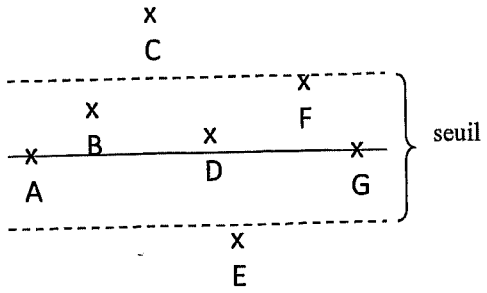
L'algorithme de Douglas-Peuker repose sur une stratégie de type « diviser pour régner ». Pour éliminer des sommets « proches de l'alignement », un seuil est fixé.

Étant donnée une ligne polygonale, le principe de l'algorithme est le suivant :

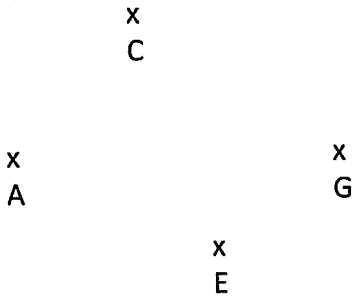
- si la ligne ne contient qu'un ou deux sommets, l'algorithme se termine ;
- sinon, on considère la droite formée par les extrémités de la ligne (son premier et dernier sommet), et on sélectionne le point le plus éloigné de cette droite (dans le cas où les extrémités sont confondues, on sélectionne le point le plus éloigné de celles-ci) :
  - si la distance entre le point sélectionné et la droite (ou les extrémités lorsqu'elles sont confondues) est inférieure au seuil fixé, on ne conserve que les extrémités de la ligne polygonale ;



- sinon, on applique l'algorithme de manière récursive aux deux parties de la ligne polygonale formées de la séquence formée du premier sommet jusqu'au sommet sélectionné d'une part et de la séquence formée du sommet sélectionné jusqu'au dernier sommet d'autre part. L'algorithme renvoie alors la concaténation des séquences simplifiées ainsi obtenues.



L'algorithme appelé sur la ligne polygonale [A,B,C,D,E,F,G] ci-dessus, va récursivement être appelé sur les lignes polygonales [A,B,C] et [C,D,E,F,G]. La ligne polygonale que l'on obtiendra à la fin de l'algorithme sera [A,C,E,G].



5. L'algorithme de Douglas-Peuker est implémenté par la fonction `simplifie` ci-dessous, qui prend en argument la ligne polygonale et le seuil choisi.

```
def simplifie(ligne, seuil):
    n = len(ligne)
    if n <= 2:
        return ...
    else:
        indice_max, dmax = le_plus_loin(ligne)

        if dmax <= seuil:
                return ...
        else:
                    #bloc à écrire
```

Recopier et compléter le code de cette fonction.

Exercice 5 (4 points)

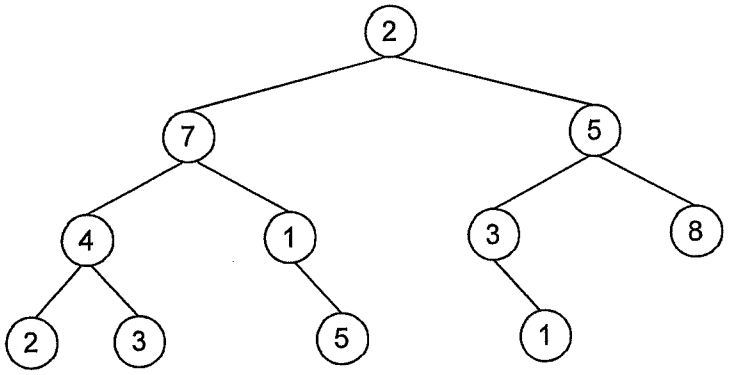
Cet exercice porte sur les arbres binaires, la programmation orienté objet et la récursivité

Dans un arbre binaire, chaque nœud admet au plus deux enfants, appelés sous-arbre gauche et sous-arbre droit. On considère dans cet exercice des arbres binaires étiquetés avec des nombres entiers.
On définit un chemin racine-feuille dans un tel arbre comme une liste ordonnée de nœuds telle que

- le premier nœud est la racine ;
- chaque nœud suivant est enfant du précédent ;
- le dernier nœud est une feuille.

On appellera somme d'un chemin racine-feuille la somme des étiquettes des nœuds du chemin.
Enfin, la plus grande somme racine-feuille d'un arbre est la plus grande somme qu'il est possible d'obtenir en considérant tous les chemins racine-feuille de l'arbre.

1. Déterminer la plus grande somme racine-feuille de l'arbre représenté ci-dessous.



2. La classe `Noeud` ci-dessous implémente le type abstrait d'arbre binaire.

```
class Noeud:

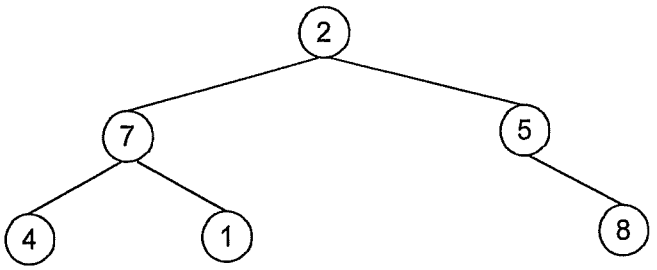
    def __init__(self, v):
        self.etiquette = v
        self.sag = None
        self.sad = None

    def niveau(self):
        if self.sag!=None and self.sad!=None:
            hg = self.sag.niveau()
            hd = self.sad.niveau()
            return 1+max(hg, hd)
        if self.sag!=None:
            return self.sag.niveau()+1
        if self.sad!=None:
            return self.sad.niveau()+1
        return 0

    def modifier_sag(self, nsag) :
        self.sag = nsag

    def modifier_sad(self, nsad) :
        self.sad = nsad
```

a. Écrire une suite d'instructions utilisant la classe `Noeud` permettant de représenter l'arbre ci-dessous.

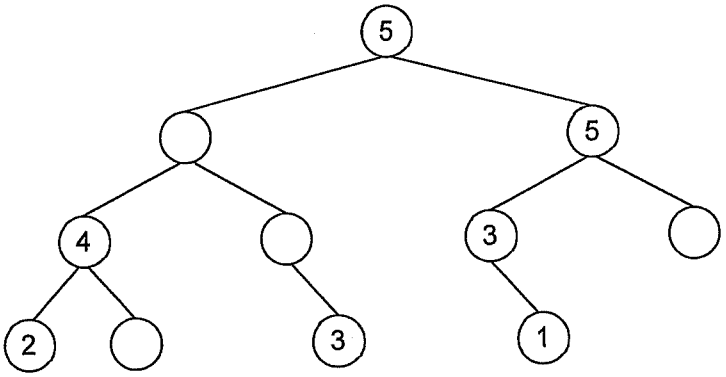


b. Que renvoie l'appel de la méthode `niveau` sur l'arbre ci-dessus ?

3. S'inspirer du code de la méthode `niveau` pour écrire une méthode récursive `pgde_somme` qui renvoie la plus grande somme racine-feuille d'un arbre.

4. On appelle arbre magique un arbre binaire dont toutes les sommes des chemins racine-feuille sont égales.

a. Recopier et compléter l'arbre ci-dessous pour qu'il soit magique.



b. Un arbre est magique si ses sous-arbres sont magiques et qu'ils ont de plus la même plus grande somme racine-feuille. Écrire une méthode récursive `est_magique` qui renvoie `True` si l'arbre est magique et `False` sinon.