BACCALAURÉAT GÉNÉRAL

ÉPREUVE D'ENSEIGNEMENT DE SPÉCIALITÉ

ÉPREUVE DE TYPE BAC
LUNDI 29 JANVIER 2024

NUMÉRIQUE ET SCIENCES INFORMATIQUES

Sujet 1

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 9 pages numérotées de 1 / 9 à 9 / 9.

Le sujet est composé de trois exercices indépendants.

EXERCICE 1 (6 points)

Cet exercice porte sur l'algorithmique, la programmation orientée objet et la méthode diviser pour régner.

L'objectif de cet exercice est de trouver les deux points les plus proches dans un nuage de points pour lesquels on connaît les coordonnées dans un repère orthogonal.
On rappelle que la distance entre deux points A et B de coordonnées $(x_A; y_A)$ et $(x_B; y_B)$ est donnée par la formule $AB = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$.
Les coordonnées d'un point seront stockées dans un tuple de deux nombres réels.
Le nuage de points sera représenté en Python par une liste de tuples de taille n, n étant le nombre total de points. On suppose qu'il n'y a pas de points confondus (mêmes abscisses et mêmes ordonnées) et qu'il y a au moins deux points dans le nuage.
Pour calculer la racine carrée, on utilisera la fonction `sqrt` du module `math`, pour rappel :

```
1 >>> from math import sqrt
2 >>> sqrt(16)
3 4.0
```

1. Cette partie comprend plusieurs questions générales :
   (a) Donner le rôle de l'instruction de la ligne 1 du code précédent.
   (b) Expliquer le résultat suivant :

```
>>> 0.1 + 0.2 == 0.3
False
```

   (c) Expliquer l'erreur suivante :

```
>>> point_A = (3, 4)
>>> point_A[0]
3
>>> point_A[0] = 2
Traceback (most recent call last):
    File "<console>", line 1, in <module>
TypeError : 'tuple' object does not support item assignment
```

2. On définit la classe `Segment` ci-dessous :

```
1 from math import sqrt
2 class Segment:
3     def __init__(self, point1, point2):
4         self.p1 = point1
5         self.p2 = point2
6         self.longueur = ..............  # à compléter
```

   (a) Recopier et compléter la ligne 6 du constructeur de la classe `Segment`.

   La fonction `liste_segments` donnée ci-dessous prend en paramètre une liste de points et renvoie une liste contenant des objets de la classe `Segment` qu'il est possible de construire à partir de ces points. On considère **les segments $[AB]$ et $[BA]$ comme étant confondus et on ajoutera un seul objet dans la liste**.

```
def liste_segments(liste_points):
    n = len(liste_points)
    segments = []
    for i in range(..............):
        for j in range(..............., n):
            # On construit le segment à partir des points i et j.
            seg = ..........................................
            segments.append(seg) # On l'ajoute à la liste
    return segments
```

(b) Recopier la fonction sans les commentaires et compléter le code manquant.

(c) Donner en fonction de `n` la longueur de la liste `segments`. Le résultat peut être laissé sous la forme d'une somme.

(d) Donner, en fonction de `n`, la complexité en temps de la fonction `liste_segments`.

3. L'objectif de cette partie est d'écrire la fonction de recherche des deux points les plus proches en utilisant la méthode diviser pour régner.

   On dispose de deux fonctions : `moitie_gauche` (respectivement `moitie_droite`) qui prennent en paramètre une liste et qui renvoient chacune une nouvelle liste contenant la moitié gauche (respectivement la moitié droite) de la liste de départ. Si le nombre d'éléments de celle-ci est impair, l'élément du centre se trouve dans la partie gauche.

   Exemples :

```
>>> liste = [1, 2, 3, 4]
>>> moitie_gauche(liste)
[1, 2]
>>> moitie_droite(liste)
[3, 4]
```

```
>>> liste = [1, 2, 3, 4, 5]
>>> moitie_gauche(liste)
[1, 2, 3]
>>> moitie_droite(liste)
[4, 5]
```

(a) Écrire la fonction `plus_court_segment` qui prend en paramètre une liste d'objets de la classe `Segment` et renvoie l'objet de la classe `Segment` dont la longueur est la plus petite.

On procédera de la façon suivante :

- Tester si le cas de base est atteint, c'est-à-dire lorsque la liste contient un seul segment ;
- Découper la liste en deux listes de tailles égales (à une unité près) ;
- Appeler récursivement la fonction pour rechercher le minimum dans chacune des deux listes ;
- Comparer les deux valeurs récupérées et renvoyer la plus petite des deux.

4. On considère les trois points $A(3;4)$, $B(2;3)$ et $C(-3;-1)$.

(a) Donner l'instruction Python permettant de construire la variable `nuage_points` contenant les trois points A, B et C.

(b) En utilisant les fonctions de l'exercice, écrire les instructions Python qui affichent les coordonnées des deux points les plus proches du nuage de points `nuage_points`.

EXERCICE 2 (6 points)

Cet exercice porte sur les arbres binaires de recherche.

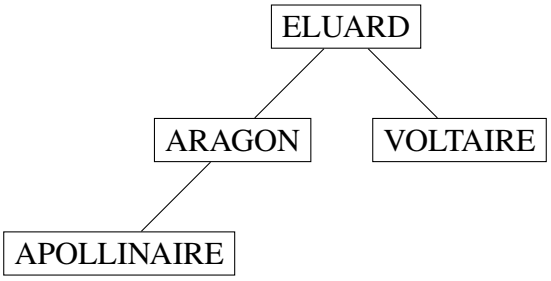
Un **arbre binaire de recherche** est un arbre binaire pour lequel chaque nœud possède une étiquette dont la valeur est supérieure ou égale à toutes les étiquettes des nœuds de son fils gauche et strictement inférieure à celles des nœuds de son fils droit.
On rappelle que :

- sa taille est son nombre de nœuds ;
- sa hauteur est le nombre de niveaux qu'il contient.

Un éditeur réédite des ouvrages. Il doit gérer un nombre important d'auteurs de la littérature. Pour stocker le nom des auteurs, il utilise un programme informatique qui les enregistre dans un arbre binaire de recherche.
L'arbre vide sera noté `Null` pour les algorithmes de cet exercice.
Si `A` est un nœud non vide, `valeur(A)` renvoie le nom de l'auteur ; `fils_gauche(A)` renvoie le fils gauche du nœud `A` et `fils_droit(A)` renvoie le fils droit du nœud `A`.
L'ordre alphabétique est utilisé pour classer le nom des auteurs.
Par exemple, on a `APOLLINAIRE < BAUDELAIRE`
Ainsi, pour tout nœud `A`, si `fils_gauche(A)` et `fils_droit(A)` ne sont pas `Null`, on a :

```
valeur(fils_gauche(A)) <= valeur(A) < valeur(fils_droit(A))
```

Par exemple, l'arbre binaire suivant `A1` est un arbre binaire de recherche :



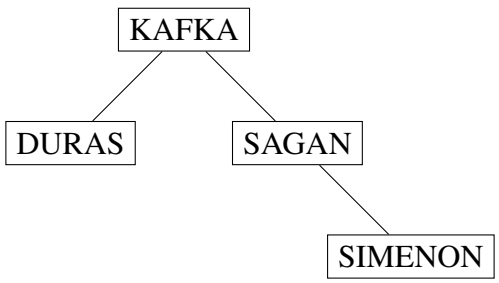
1. (a) Recopier et compléter l'arbre binaire de recherche précédent en insérant successivement dans cet ordre les noms suivants :

   `DUMAS ; HUGO ; ZWEIG ; ZOLA`

   (b) Quelle est la taille de l'arbre obtenu ? Quelle est la hauteur de cet arbre ?

   (c) Plus généralement, si l'arbre est de hauteur h, quel est le nombre maximal d'auteurs enregistrés dans cet arbre en fonction de h ?

On définit ici l'équilibre d'un arbre binaire : il s'agit d'un nombre entier positif ou négatif. Il vaut 0 si l'arbre est vide. Sinon il vaut la différence des hauteurs des sous-arbres gauche et droit de l'arbre.
Par exemple, si on considère l'arbre suivant que l'on nommera `A2` :



Son équilibre vaut -1 car :
la hauteur de son sous-arbre gauche vaut 1, la hauteur de son sous-arbre droit vaut 2 et $1 - 2 = -1$.
Un arbre est dit équilibré si son équilibre vaut -1, 0 ou 1.
L'arbre précédent est donc équilibré.

2. Recopier et compléter l'arbre de ce dernier exemple avec les noms FLAUBERT, BALZAC, PROUST, SAND, WOOLF, COLETTE, CHRISTIE et AUDIARD quitte à modifier l'ordre d'insertion de manière à ce que cet arbre reste équilibré.

L'éditeur souhaite utiliser une fonction récursive `cherche_a(ABR, NOM)` qui prend en paramètres ABR, un arbre binaire de recherche, et NOM, un nom d'auteur. La fonction renvoie VRAI si NOM est une étiquette de l'arbre ABR et FAUX dans le cas contraire.
On considère une première version de la fonction ci-dessous :

```
Fonction cherche_a(ABR, t) :
  SI ABR = NULL :
    RENVOYER FAUX
  SINON SI valeur(ABR) = t :
    RENVOYER VRAI
  SINON :
    RENVOYER cherche_a(fils_gauche(ABR),t) OU cherche_a(fils_droit(ABR),t)
```

3. (a) Que renvoie l'appel `cherche_a(A2,'SIMENON')` ? Justifier la réponse.
   (b) Quelle est la complexité de l'algorithme donné en fonction du nombre de nœuds ?
   (c) Corriger la définition de la fonction `cherche_a(ABR, NOM)` en la rendant plus efficace, autrement dit en faisant en sorte qu'elle tire profit des caractéristiques particulières d'un arbre binaire de recherche.
   (d) Préciser la complexité de l'algorithme corrigé en fonction du nombre de nœuds.

L'éditeur souhaite utiliser une fonction récursive `hauteur(ABR)` qui prend en paramètre un arbre binaire ABR et renvoie la hauteur de cet arbre.

4. Écrire un algorithme de la fonction `hauteur(ABR)` qui prend en entrée ABR, un arbre binaire de recherche, et renvoie sa hauteur. On pourra avoir recours aux fonctions `MIN(val1, val2)` ou `MAX(val1, val2)` qui renvoient respectivement la plus petite et la plus grande valeur entre deux nombres `val1` et `val2`.
5. Rappeler quel type de parcours en profondeur d'abord de l'arbre permet d'obtenir ses valeurs par ordre croissant.

EXERCICE 3 (8 points)

Cet exercice porte sur la programmation Python (dictionnaire), la programmation orientée objet, les bases de données relationnelles et les requêtes SQL.

Cet exercice est composé de 3 parties indépendantes.

On veut créer une application permettant de stocker et de traiter des informations sur des livres de science-fiction. On désire stocker les informations suivantes :

- l'identifiant du livre (`id`) ;
- le titre (`titre`) ;
- le nom de l'auteur (`nom_auteur`) ;
- l'année de première publication (`ann_pub`) ;
- une note sur 10 (`note`).

Voici un extrait des informations que l'on cherche à stocker :

Livres de science-fiction				
id	titre	auteur	ann_pub	note
1	1984	Orwell	1949	10
2	Dune	Herbert	1965	8
14	Fondation	Asimov	1951	9
4	Ubik	K.Dick	1953	9
8	Blade Runner	K.Dick	1968	8
7	Les Robots	Asimov	1950	10
15	Ravage	Barjavel	1943	6
17	Chroniques martiennes	Bradbury	1950	7
9	Dragon déchu	Hamilton	2003	8
10	Fahrenheit 451	Bradbury	1953	8

Partie A

Dans cette première partie, on utilise un dictionnaire Python. On considère le programme suivant :

```
dico_livres = {
    'id' : [1, 2, 14, 4, 5, 8, 7, 15, 9, 10],
    'titre' : ['1984', 'Dune', 'Fondation', 'Ubik', 'Blade Runner',
        'Les Robots', 'Ravage', 'Chroniques martiennes',
        'Dragon déchu', 'Fahrenheit 451'],
    'auteur' : ['Orwell', 'Herbert', 'Asimov',
        'K.Dick', 'K.Dick', 'Asimov', 'Barjavel',
        'Bradbury', 'Hamilton', 'Bradbury'],
    'ann_pub' : [1949, 1965, 1951, 1953,1968,
        1950, 1943, 1950, 2003, 1953],
    'note' : [10, 8, 9, 9, 8, 10, 6, 7, 8, 8]
}
a = dico_livres['note']
b = dico_livres['titre'][2]
```

1. Déterminer les valeurs des variables `a` et `b` après l'exécution de ce programme.

La fonction `titre_livre` prend en paramètre un dictionnaire (de même structure que `dico_livres`) et un identifiant, et renvoie le titre du livre qui correspond à cet identifiant. Dans le cas où l'identifiant passé en paramètre n'est pas présent dans le dictionnaire, la fonction renvoie **`None`**.

```
1  def titre_livre(dico, id_livre):
2      for i in range(len(dico['id'])):
3          if dico['id'][i] == ... :
4              return dico['titre'][...]
5      return ...
```

2. Recopier et compléter les lignes 3, 4 et 5 de la fonction `titre_livre`.
3. Écrire une fonction `note_maxi` qui prend en paramètre un dictionnaire `dico` (de même structure que `dico_livres`) et qui renvoie la note maximale.
4. Écrire une fonction `livres_note` qui prend en paramètre un dictionnaire `dico` (de même structure que `dico_livres`) et une note `n`, et qui renvoie la liste des titres des livres ayant obtenu la note `n` (on rappelle que `t.append(a)` permet de rajouter l'élément `a` à la fin de la liste `t`).
5. Écrire une fonction `livre_note_maxi` qui prend en paramètre un dictionnaire `dico` (de même structure que `dico_livres`) et qui renvoie la liste des titres des livres ayant obtenu la meilleure note sous la forme d'une liste Python.

Partie B

Dans cette partie, on utilise le paradigme de la programmation orientée objet (POO).
On propose deux classes : `Livre` et `Bibliotheque`.

```
class Livre:
    def __init__(self, id_livre, titre, auteur, ann_pub, note):
        self.id = id_livre
        self.titre = titre
        self.auteur = auteur
        self.ann_pub = ann_pub
        self.note = note
    def get_id(self):
        return self.id
    def get_titre(self):
        return self.titre
    def get_auteur(self):
        return self.auteur
    def get_ann_pub(self):
        return self.ann_pub


class Bibliotheque:
    def __init__(self):
        self.liste_livre = []
    def ajout_livre(self, livre):
        self.liste_livre.append(livre)
```

```
    def titre_livre(self, id_livre):
        for livre in self.liste_livre :
            if ... == id_livre :
                return ...
        return ...
```

6. Citer un attribut et une méthode de la classe `Livre`.
7. Écrire la méthode `get_note` de la classe `Livre`. Cette méthode devra renvoyer la note d'un livre.
8. Recopier et compléter la méthode `titre_livre` de la classe `Bibliotheque`.
   Cette méthode prend en paramètre l'identifiant d'un livre et renvoie le titre du livre si l'identifiant existe, ou `None` si l'identifiant n'existe pas.
9. Écrire le programme permettant de créer une bibliothèque nommée `biblio` et d'y ajouter le livre Blade Runner (voir le tableau en début d'exercice) en utilisant les classes `Livre` et `Bibliotheque`.

Partie C

On utilise maintenant une base de données relationnelle. Les commandes nécessaires ont été exécutées afin de créer une table `livres`. Cette table `livres` contient toutes les données sur les livres. On obtient donc la table donnée au tout début de l'exercice.
L'attribut `id` est la clé primaire pour la table `livres`.

10. Expliquer pourquoi l'attribut `auteur` ne peut pas être choisi comme clé primaire.
11. Décrire puis donner le résultat renvoyé par la requête SQL suivante :

```
SELECT titre
FROM livres
WHERE auteur = 'K.Dick';
```

12. Écrire une requête SQL permettant d'obtenir les titres des livres écrits par Asimov publiés après 1950.
13. Écrire une requête SQL permettant de modifier la note du livre Ubik en la passant de 9/10 à 10/10.
14. Écrire une requête SQL donnant le nombre de titres dont la note est supérieure ou égale à 9.

On souhaite proposer plus d'informations sur les auteurs des livres. Pour cela, on crée une deuxième table `auteurs` avec les attributs suivants :

- `id` de type `INT` ;
- `nom` de type `TEXT` ;
- `prenom` de type `TEXT` ;
- `annee_naissance` de type `INT` (année de naissance).

auteurs			
id	nom	prenom	annee_naissance
1	Orwell	George	1903
2	Herbert	Franck	1920
3	Asimov	Isaac	1920
4	K.Dick	Philip	1928
5	Bradbury	Ray	1920
6	Barjavel	René	1911
7	Hamilton	Peter	1960

La table `livres` est aussi modifiée comme suit :

livres				
id	titre	id_auteur	ann_pub	note
1	1984	1	1949	10
2	Dune	2	1965	8
14	Fondation	3	1951	9
4	Ubik	4	1953	9
8	Blade Runner	4	1968	8
7	Les Robots	3	1950	10
15	Ravage	6	1943	6
17	Chroniques martiennes	5	1950	7
9	Dragon déchu	7	2003	8
10	Fahrenheit 451	5	1953	8

15. Expliquer l'intérêt d'utiliser deux tables (`livres` et `auteurs`) au lieu de regrouper toutes les informations dans une seule table.
16. Expliquer le rôle de l'attribut `id_auteur` de la table `livres`.
17. Écrire une requête SQL qui renvoie le nom et le prénom des auteurs des livres publiés après 1960.
18. Décrire par une phrase en français le résultat de la requête SQL suivante :

```sql
SELECT titre
FROM livres
JOIN auteurs ON id_auteur = auteurs.id
WHERE ann_pub - annee_naissance < 30;
```