Lycée Les 3 Sources **Bourg-Lès-Valence**
NSI T^le Année **2025-26**

Devoir surveillé n°4 – Correction

Nom et prénom :

Exercice 1 : (13pt) *Cet exercice porte sur les arbres binaires et la programmation Python.*
Le codage de Shannon-Fano est un système de codage utilisé pour la compression sans pertes de données. Il a été mis au point par Robert Fano d'après une idée de Claude Shannon.

Partie A

Dans cette partie, on va étudier l'utilisation des arbres de codage.
Un arbre de codage est un arbre binaire où chaque feuille contient un symbole du texte que l'on souhaite coder. Le code binaire d'un symbole s'obtient alors en concaténant les `0` et les `1` sur les branches qui mènent de la racine à la feuille contenant ce symbole. Par exemple, pour l'arbre de codage donné en Figure 1, le symbole `c` est codé par le mot binaire `1101`, tandis que `d` est codé par le mot binaire `11000`. Les codes binaires des symboles ne sont donc pas tous de la même taille. Pour décoder un mot binaire, il suffit de descendre dans l'arbre, depuis la racine, selon les `0` et les `1` qu'on lit jusqu'à trouver une feuille (et donc un symbole), puis de recommencer avec la suite du mot binaire pour décoder les symboles suivants.

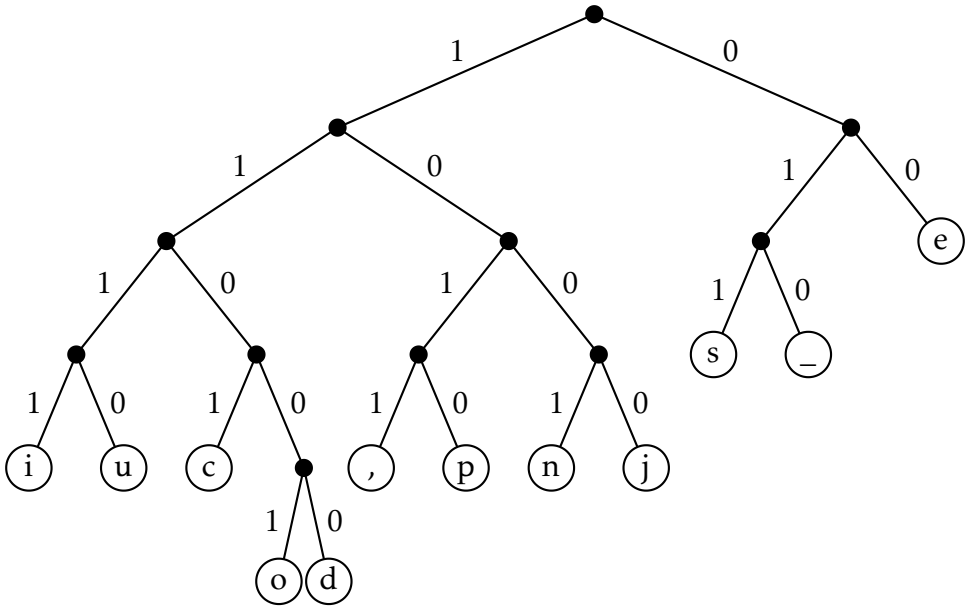


Figure 1. Exemple d'arbre de codage

1) Écrire le mot binaire qui sera utilisé pour encoder le caractère espace, représenté par le symbole _ dans l'arbre.
   Solution : Le mot sera `010`.
2) Déterminer le texte codé par le mot binaire `000111010111111001100 1`.
   Solution : On trouve $\underbrace{00}_{e}\ \underbrace{011}_{s}\ \underbrace{1010}_{p}\ \underbrace{1111}_{i}\ \underbrace{11001}_{o}\ \underbrace{1001}_{n}$.
3) Citer le type de parcours de l'arbre qui permettrait d'obtenir les symboles classés par taille d'encodage croissante.
   Solution : Il faut faire un parcours en largeur.

Partie B

Dans cette partie, on va utiliser le codage de Shannon-Fano pour encoder le texte :
`je pense, donc je suis`

Dans la méthode de Shannon-Fano, l'arbre de codage est calculé pour un texte donné par l'algorithme suivant.

- Étape 1 : classer les symboles du texte par nombre d'occurrences croissant ;
- Étape 2 : en gardant le classement obtenu, séparer les symboles en deux sous-groupes de sorte que les totaux des nombres d'occurrences soient les plus proches possibles dans les deux sous-groupes ;
- Étape 3 : placer tous les symboles du premier groupe dans le fils gauche (côté étiqueté par 1), et ceux du second groupe dans le fils droit (côté étiqueté par `0`) ;
- Étape 4 : recommencer récursivement pour chacun des sous-groupes jusqu'à ce qu'ils n'aient plus qu'un seul symbole ; on a alors une feuille étiquetée par ce symbole.

Après avoir classé les symboles par nombre d'occurrences croissant (étape 1), on obtient le tableau suivant :

symbole	i	u	c	o	d	,	p	n	j	s	_	e
nombre d'occurrences	1	1	1	1	1	1	1	2	2	3	4	4

4) Justifier par le calcul que l'étape 2 mène à la situation illustrée par la Figure 2.
Solution : Dans la partie gauche du tableau, on a un total de 11 occurrences, tout comme dans la partie droite.

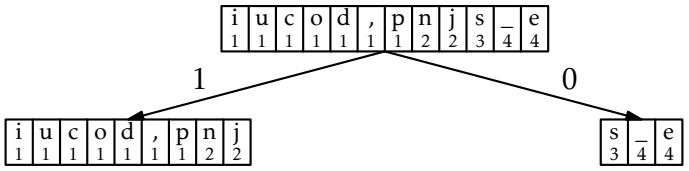


Figure 2. Le résultat de l'étape 2

En appliquant l'algorithme de Shannon-Fano, on peut obtenir l'arbre de la Figure 3.

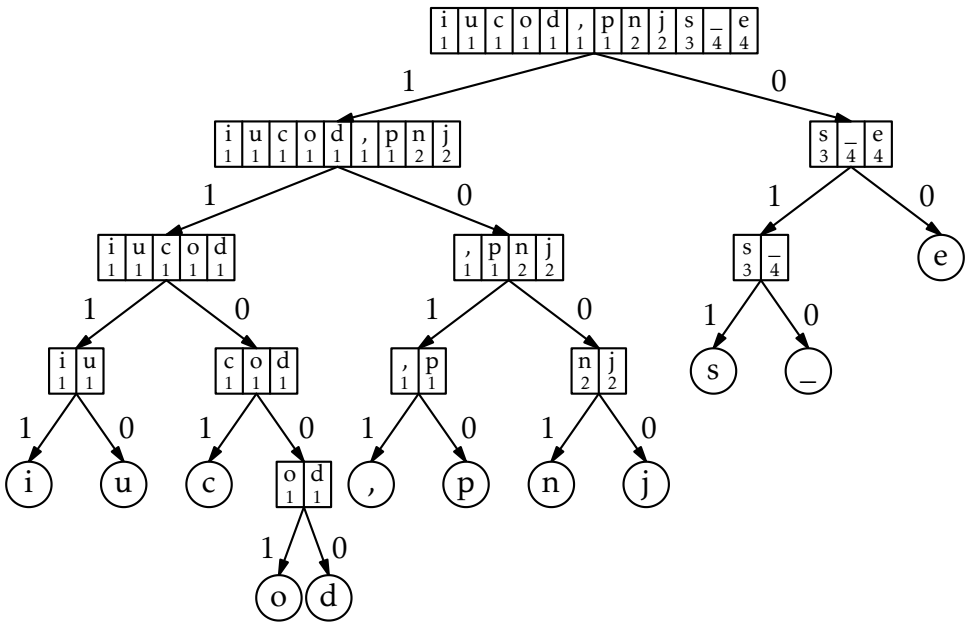


Figure 3. Arbre de codage obtenu par l'algorithme de Shannon-Fano

Nom et prénom :

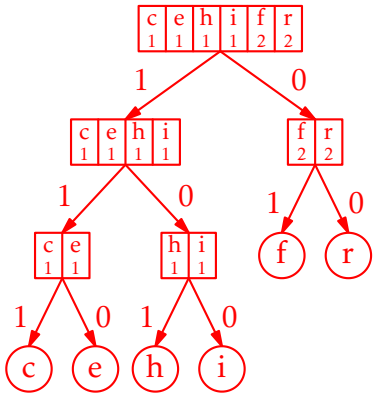
On rappelle qu'un arbre réduit à un seul nœud, c'est-à-dire réduit à une feuille, est de hauteur 0.

5) Donner la hauteur de l'arbre de la Figure 3 et préciser dans le contexte de l'exercice ce qu'elle représente.
Solution : La hauteur est de 5.

On rappelle que dans le code ASCII, chaque symbole est codé sur un octet.

6) Justifier, en comparant le codage ASCII et le codage de Shannon-Fano, que ce second codage permet d'utiliser environ deux fois moins d'octets pour le texte :
`je pense, donc je suis`
Solution : Pour le codage ASCII, il faut 22 octets, donc 176 bits. Avec le codage de Shanon-Fano, on obtient une chaîne de 75 bits. C'est donc plus de deux fois moins.

7) Dessiner, en vous inspirant de l'arbre de la Figure 1, un arbre de codage qui permettrait d'encoder le mot « `chiffrer` » en utilisant l'algorithme de Shannon-Fano.



Partie C

Dans cette partie, on souhaite écrire une fonction Python qui donnera le mot binaire obtenu pour coder un texte avec l'algorithme de Shannon-Fano. On commence par la fonction `creer_dico_occ` :

```
1  def creer_dico_occ(texte):
2      """renvoie un dictionnaire dont les clés sont les
3      symboles de texte et les valeurs associées leur
4      nombre d'occurences dans texte"""
5      dico = {}
6      for symbole in texte:
7          if symbole in dico:
8              dico[symbole] = ...
9          else:
10             dico[symbole] = ...
11     return dico
```

8) Compléter les lignes 8 et 10 du code de la fonction `creer_dico_occ`.

```
def creer_dico_occ(texte):
    dico = {}
    for symbole in texte:
        if symbole in dico:
            dico[symbole] = dico[symbole] + 1
        else:
            dico[symbole] = 1
    return dico
```

On dispose d'une fonction `creer_tab_trie` qui prend en paramètre un dictionnaire construit avec la fonction `creer_dico_occ` et qui renvoie une liste de tuples classés dans l'ordre croissant d'occurrences des symboles. Par exemple :

```
>>> texte = 'je pense, donc je suis'
>>> dico = creer_dico_occ(texte)
>>> creer_tab_trie(dico)
[('i', 1), ('u', 1), ('c', 1), ('o', 1), ('d', 1), (',', 1),
 ('p', 1), ('n', 2), ('j', 2), ('s', 3), (' ', 4), ('e', 4)]
```

9) Écrire une fonction `somme_occ` qui prend en paramètres un tableau `tab` de tuples `(symbole, nb_occ)` et qui renvoie la somme des nombres d'occurrences des symboles du tableau. Les tuples utilisés sont de même structure que l'élément renvoyé dans l'exemple précédent.

```
def somme_occ(tab):
    somme = 0
    for symbole, nb_occ in tab:
        somme += nb_occ
    return somme
```

On suppose pour la suite qu'on dispose d'une fonction separe qui sépare un tableau trié en deux sous-tableaux de manière à ce que les sommes de ces derniers soient les plus proches possible :

```
1  def separe(tab):
2      moitie = somme_occ(tab) // 2
3      somme = 0
4      i = 0
5      while moitie > somme:
6          somme = somme + tab[i][1]
7          i = i + 1
8          tab1 = [tab[k] for k in range(0, i)]
9          tab2 = [tab[k] for k in range(i, len(tab))]
10     return tab1, tab2
```

10) Compléter les lignes 9 et 11 du code de la fonction récursive `shannon` qui prend en paramètres un caractère `symbole` et un tableau trié `tab` et qui renvoie l'écriture binaire associée à symbole dans le tableau `tab`.

```
1  def shannon(symbole, tab):
2      """renvoie l'écriture binaire associée à symbole
3      dans le tableau trié tab"""
4      if len(tab) == 1:
5          return ""
6      else:
7          t1, t2 = separe(tab)
8          if symbole in [elt[0] for elt in t1]:
9              return "1" + ...
10         else:
11             return "0" + ...
```

```
def shannon(symbole, tab):
    if len(tab) == 1:
        return ""
    else:
        t1, t2 = separe(tab)
        if symbole in [elt[0] for elt in t1]:
            return "1" + shannon(symbole, t1)
        else:
            return "0" + shannon(symbole, t2)
```

11) Décrire ce qui garantit la terminaison de la fonction récursive shannon.
Solution : Lors de chaque appel récursif, la taille du tableau est divisé par 2. Donc il finira forcément par être vide.

12) Écrire une fonction encode_shannon qui prend en paramètre un texte de type **str** et renvoie un mot binaire de type **str** obtenu après encodage par l'algorithme de Shannon-Fano.
On pourra utiliser les fonctions vues précédemment qui sont recensées ci-après.

```
creer_dico_occ(texte)
  renvoie un dictionnaire dont les clés sont les symboles
  du texte et les valeurs associées leur nombre
  d'occurrences

creer_tab_trie(dico)
  renvoie la liste crée à partir d'un dictionnaire
  de couples (symbole, nb_occ)

separe(tab)
  renvoie le tuple composé des 2 sous-tableaux triés
  avec des sommes d'occurences proches

shannon(symbole, tab)
  renvoie l'écriture binaire associée au symbole dans le
  tableau trié tab
```

```
def encode_shannon(texte):
    dico = creer_dico_occ(texte)
    tab = creer_tab_trie(dico)
    reponse = ""
    for s in texte:
        reponse = reopnse + shannon(s, tab)
    return reponse
```

Exercice 2 : (14pt) *Cet exercice porte sur les bases de données et les requêtes SQL, les arbres binaires et les algorithmes sur les arbres binaires.*

Partie A

Dans cet exercice, on pourra utiliser les clauses du langage SQL pour :

- construire des requêtes d'interrogation à l'aide de **`SELECT`**, **`FROM`**, **`WHERE`** (avec les opérateurs logiques **`AND`**, **`OR`**), **`JOIN`** ... **`ON`** ;
- construire des requêtes d'insertion et de mise à jour à l'aide de **`UPDATE`**, **`INSERT`**, **`DELETE`**.

Une exoplanète est une planète située hors du système solaire. La plupart des exoplanètes découvertes à ce jour orbitent autour d'une unique étoile.
Une étoile est repérée précisément dans le ciel par son ascension droite et sa déclinaison (voir Figure 1). La direction de coordonnées `(0, 0)` est une direction fixe du ciel servant d'origine de ce système de coordonnées.

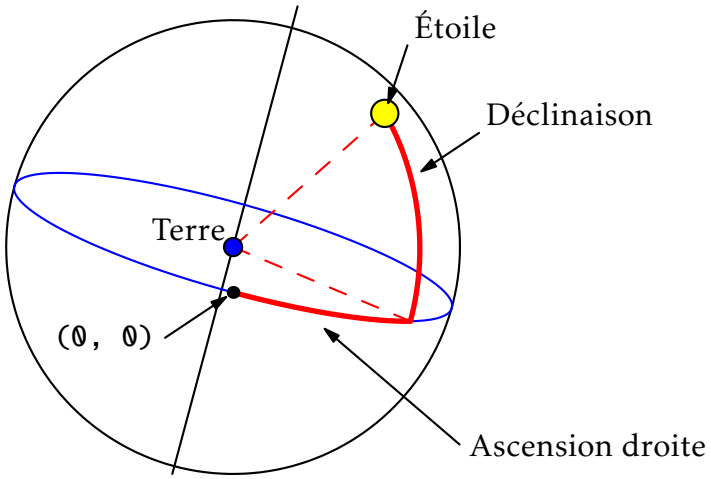


Figure 1. Coordonnées d'une étoile

On considère dans cet exercice deux relations décrivant des étoiles et les exoplanètes orbitant autour d'elles :

- la relation `Etoiles` contient les informations décrivant des étoiles :
  - `id_etoile` : l'identifiant unique de l'étoile (nombre entier) ;
  - `nom` : le nom de l'étoile (chaîne de caractères) ;
  - `ascension` : l'ascension droite de l'étoile en degré (nombre réel) ;
  - `declinaison` : la déclinaison de l'étoile en degré (nombre réel).
- la relation `Exoplanetes` contient les informations décrivant des exoplanètes :
  - `id_exoplanete` : l'identifiant unique de l'exoplanète (nombre entier) ;
  - `masse` : la masse de l'exoplanète, exprimée sous la forme d'une fraction de la masse de la planète Jupiter (nombre réel) ;
  - `rayon` : le rayon de l'exoplanète, exprimée sous la forme d'une fraction du rayon de la planète Jupiter (nombre réel) ;
  - `id_etoile` : l'identifiant de l'étoile autour de laquelle orbite l'exoplanète (nombre entier).

Une exoplanète dont l'attribut `masse` est égal à `6.84` a une masse 6,84 fois plus grande que celle de la planète Jupiter.

On fournit ci-dessous des extraits de ces deux tables :

Etoiles			
id_etoile	nom	ascension	declinaison
1	109 Psc	26.23	20.08
2	beta Pic	86.82	−51.07
3	K2-21	340.30	−14.49
4	Kepler-11	297.12	41.91

Exoplanetes			
id_exoplanete	masse	rayon	id_etoile
1	6.84	1.15	1
2	11.90	1.65	2
3	8.89	1.20	2
4	0.01	0.16	3
5	0.02	0.22	3
6	0.01	0.16	4
7	0.01	0.26	4

L'attribut `id_exoplanete` est la clé primaire de la relation `Exoplanetes`. L'attribut `id_etoile` est la clé primaire de la relation `Etoiles`.

1) Expliquer pourquoi l'attribut `masse` de la relation `Exoplanetes` ne peut pas servir de clé primaire de cette relation.
Solution : Il y a plusieurs planètes avec la même masse. Les valeurs ne sont pas uniques et cet attribut ne peut donc pas être une clé primaire.
2) Donner le nom de l'attribut pouvant être utilisé comme clé étrangère dans la relation `Exoplanetes`. Expliquer son rôle.
Solution : On peut utiliser `id_etoile` pour relier la table `Exoplanetes` à la table `Etoile`.

3) Donner le résultat de la requête SQL suivante :

```
SELECT masse, rayon
FROM Exoplanetes
WHERE id_exoplanete = 4;
```

Solution : On va obtenir la masse et le rayon de l'exoplanète dont l'identifiant est 4. On va donc obtenir 0,01 et 0,16.
4) Écrire une requête SQL permettant d'obtenir l'identifiant et le nom des étoiles dont l'ascension droite est supérieure ou égale à 100 degrés.
Solution :

```
SELECT id_etoile, nom FROM Etoiles
WHERE ascension >= 100;
```

On souhaite insérer une nouvelle exoplanète de rayon égal à 0,37 fois celui de Jupiter et pesant 0,03 fois la masse de Jupiter. Cette exoplanète orbite autour de l'étoile Kepler-11 dont l'identifiant est 4. On pourra attribuer à cette nouvelle exoplanète l'identifiant 9 qui n'apparaît pas dans la relation `Exoplanetes`.
5) Écrire une requête SQL permettant d'insérer cette nouvelle exoplanète dans la base de données.
Solution :

```
INSERT INTO Exoplanetes VALUES (9, 0.03, 0.37, 4);
```

6) Écrire une requête SQL permettant d'obtenir les rayons des exoplanètes orbitant autour de l'étoile nommée Kepler-11, dont l'identifiant est supposé non connu.

Solution :

```
SELECT rayon FROM Exoplanetes
JOIN Etoiles ON Etoiles.id_etoile = Exoplanetes.id_etoile
WHERE nom = "Kepler-11";
```

Partie B

On souhaite désormais écrire une application Python permettant de classer et de retrouver efficacement les étoiles selon leur position dans le ciel.

On rappelle qu'une étoile est repérée par son ascension droite et sa déclinaison. Par souci de simplicité, on considère désormais que deux étoiles ont toujours des coordonnées entières et distinctes. On représente en Python les coordonnées d'une étoile par un tuple d'entiers `(ascension, declinaison)`.

Dans la suite, on considère les étoiles dont les coordonnées sont contenues dans la liste de tuples etoiles définie par :

```
etoiles = [(29, 21), (17, 14), (10, 30), (35, 13), (30, 63), (15, 20)]
```

On cherche à construire un arbre binaire de recherche à partir des coordonnées présentes dans la liste `etoiles` afin d'accélérer les opérations de traitement sur celles-ci. Pour cela :

- on commence par trier la liste `etoiles` par ordre croissant, afin que l'arbre résultant soit de hauteur minimale ;
- pour construire l'arbre binaire de recherche à partir des éléments de la liste `etoiles` compris entre les indices `debut` (inclu) et `fin` (exclu) :
  - la racine de l'arbre est l'élément d'indice `milieu` définit par `milieu = (debut + fin)//2` ;
  - on construit récursivement le sous arbre gauche à l'aide des éléments de la liste `etoiles` compris entre les indices `debut` (inclu) et `milieu` (exclu) ;
  - on construit récursivement le sous arbre droit à l'aide des éléments de la liste `etoiles` compris entre les indices `milieu + 1` (inclu) et `fin` (exclu).

Pour implémenter cet algorithme, on représente en Python les arbres binaires non vides à l'aide de tuples de trois éléments `(sag, position, sad)` dans lesquels :

- `position` est la valeur de la racine. Cette valeur est le couple de coordonnées permettant de repérer l'étoile ;
- `sag` et `sad` sont respectivement les sous-arbres gauche et droit de l'arbre.

L'arbre vide est quant à lui représenté par **`None`**.

On rappelle que l'on peut comparer des tuples en Python à l'aide de l'opérateur `<` : on compare tout d'abord les valeurs à l'indice 0 de chaque couple puis, en cas d'égalité, celles à l'indice 1.

Ainsi, les expressions `(1, 4) < (2, 3)` et `(1, 4) < (1, 6)` s'évaluent toutes les deux à **`True`**.

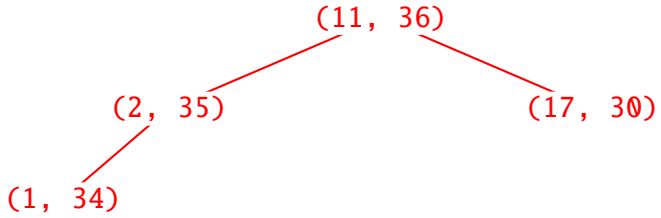
La fonction **`sorted`** de Python prend en argument une liste et renvoie une nouvelle liste contenant les mêmes valeurs triées dans l'ordre croissant à l'aide de l'opérateur `<`.

7) Donner la liste renvoyée par l'instruction **`sorted`**`(etoiles)`.

Solution : `[(10, 30), (15, 20), (17, 14), (29, 21), (30, 63), (35, 13)]`

8) Dessiner l'arbre binaire représenté par le tuple :

```
(((None, (1, 34), None), (2, 35), None), (11, 36), (None, (17, 30), None))
```



L'arbre construit à partir de la liste etoiles a donc pour représentation Python :

```
(((None, (10, 30), None), (15, 20), (None, (17, 14), None)),
 (29, 21), ((None, (30, 63), None), (35, 13), None))
```

Il est représenté sur la Figure 2 ci-après.

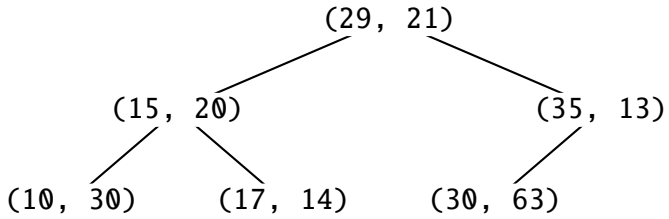
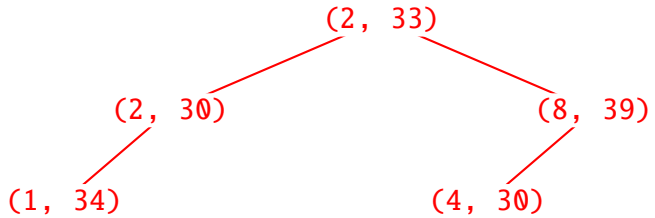


Figure 2. Arbre associé à la liste `etoiles`

9) Dessiner l'arbre binaire de recherche obtenu à partir de la liste :

```
[(1, 33), (2, 30), (2, 33), (4, 30), (8, 39)]
```



10) Compléter les lignes 3, 7, 8, 9 et 11 du code de la fonction `construction` qui prend en paramètres une liste `etoiles` supposée triée par ordre croissant, ainsi que deux entiers `debut` et `fin`. Cette fonction renverra l'arbre binaire de recherche associé aux coordonnées présentes entre les indices `debut` (inclus) et `fin` (exclu) de la liste `etoiles`.
Par exemple, l'appel initial permettant de construire l'arbre associé à la liste `etoiles` est `construction(etoiles, 0, 6)`.
L'indice du milieu est 3, le sous-arbre gauche est renvoyé par l'appel `construction(etoiles, 0, 3)` et le sous-arbre droit par `construction(etoiles, 4, 6)`.

```
1  def construction(etoiles, debut, fin):
2      if debut == fin:
3          return ...
4
5      milieu = (debut + fin) // 2
6
7      sag = construction(.....................................)
8      racine = ...
9      sad = ...
10
11     return ...
```

```
def construction(etoiles, debut, fin):
    if debut == fin:
        return None

    milieu = (debut + fin) // 2

    sag = construction(etoiles, debut, milieu)
    racine = etoiles[milieu]
    sad = construction(etoiles, milieu+1, fin)

    return (sag, racine, sad)
```

11) Écrire le code de la fonction `en_arbre` qui prend en paramètre une liste `etoiles` de couples de coordonnées non triés et renvoie l'arbre construit selon la démarche décrite plus haut. On pourra utiliser la fonction `construction` de la question précédente.

```
def en_arbre(etoiles):
    liste = sorted(etoiles)
    return construction(liste, 0, len(liste))
```

On souhaite désormais écrire une fonction `contient` qui prend en paramètres un arbre binaire de recherche `arbre` tel que renvoyé par la fonction `construction` ainsi qu'un tuple d'entiers `position` représentant les coordonnées d'une étoile. Cette fonction renvoie **`True`** si l'arbre contient cette étoile, **`False`** dans le cas contraire.

12) Compléter les lignes 3, 8, 9, 10 et 12 du code de la fonction `contient`.

```
1  def contient(arbre, position):
2      if arbre is None:
3          return ...
4
5      sag, valeur, sad = arbre
6
7      if position < valeur:
8          return contient(....................., .....................)
9      elif ..............................:
10         return ...
11     else:
12         return ...
```

```
def contient(arbre, position):
    if arbre is None:
        return False

    sag, valeur, sad = arbre

    if position < valeur:
        return contient(arbre[0], position)
    elif position > valeur:
        return contient(arbre[2], position)
    else:
        return True
```