Cet exercice porte sur les arbres binaires de recherche, la programmation orientée objet et la récursivité.

Le code Morse doit son nom à Samuel Morse, l'un des inventeurs du télégraphe.
Il a été conçu pour transférer rapidement des messages en utilisant une série de points et de tirets.
Pour cet exercice, les points seront représentés par le caractère "o" et les tirets par le caractère "-".

Chaque caractère du message que l'on veut transmettre est constitué d'une série de 1 à 5 points ou tirets. Le code a été conçu en tenant compte de la fréquence de chaque caractère dans la langue anglaise, de sorte que les caractères les plus fréquents, tels que E et T, ne comportent qu'un seul point ou tiret (E = "o", T = "-"), tandis que les caractères moins fréquents peuvent comporter 4 à 5 points ou tirets (par exemple, Q = "- - o -" et J = "o - - -").

Pour connaître le code morse de chaque caractère, on peut utiliser l'arbre binaire ci-dessous. En partant de la racine de l'arbre, la succession des branches reliant le nœud racine au caractère recherché nous donne le code morse de ce caractère en considérant que :

- une branche gauche correspond à un point ("o") ;
- une branche droite correspond à un tiret ("-").

Par exemple, le code morse de la lettre P est "**o - - o**" comme expliqué sur ce schéma :

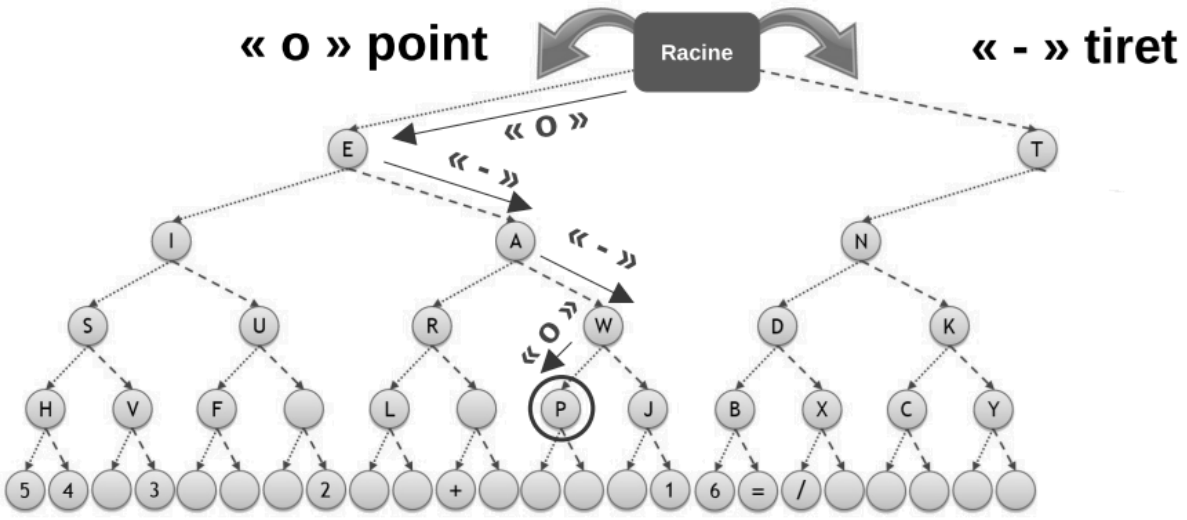

Figure 1 : Extrait de l'arbre binaire du code Morse

1. Déterminer le code morse du message "NSI", à l'aide de la figure de l'arbre binaire, en laissant un espace entre le code de chaque lettre.

2. Représenter le sous-arbre binaire pour les lettres M, G, O, Z et Q à l'aide de l'extrait de la table du code morse international :

G : --o **M :** -- **O :** --- **Q :** --o- **Z :** --oo

On donne, la déclaration de la classe et un extrait de la définition de l'arbre binaire :

```
1.  class Noeud:
2.    def __init__(self, valeur, gauche=None, droite=None):
3.      self.valeur = valeur
4.      self.gauche = gauche
5.      self.droite = droite
6.
7.  arbre = Noeud("Racine")
8.  arbre.gauche = Noeud("E")
9.  arbre.droite = Noeud("T")
10. arbre.gauche.gauche = Noeud("I")
11. arbre.gauche.droite = Noeud("A")
12. arbre.droite.gauche = Noeud("N")
13. arbre.droite.droite = Noeud("M")
```

3. Écrire les instructions à placer en ligne 14 et 15 permettant de créer les nœuds pour les lettres K et S.

4. La fonction `est_present(n,car)` permet de tester si le caractère `car` est présent ou non dans l'arbre `n` de type `Noeud`.

```
1.   def est_present(n, car) :
2.       if n == ..... :
3.           return False
4.       elif n.valeur == ..... :
5.           return True
6.       else :
7.           return est_present(n.droite, car) or .....
```

   a. Recopier le code et compléter les lignes 2, 4 et 7 de la fonction `est_present`.

   b. La fonction `est_present` est-elle récursive ? Justifier votre réponse.

   c. Déterminer quel type de parcours utilise la fonction `est_present`.

5. La fonction `code_morse(n, car)` permet de traduire un caractère `car` **présent** dans l'arbre `n` et renvoie son code morse sous forme d'une chaîne de caractères.

```
8.   def code_morse(n, car):
9.       if n.valeur == car :
10.          return .....
```

```
11.      elif est_present(.....) :
12.          return "-" + code_morse(n.droite, car)
13.      else :
14.          return .....
```

a. Recopier et compléter les ..... des lignes 10, 11 et 14 de la fonction `code_morse`.

b. Écrire une fonction `morse_message` qui reçoit un arbre de code morse et un message sous forme d'une chaîne de caractères et renvoie le message codé où chaque lettre est séparée par un trait vertical. Par exemple :

```
>>> morse_message(arbre, 'PYTHON')
>>>  o--o|-o--|-|oooo|---|-o|
```