Mini projet 5 : déchiffrer un code avec un arbre binaire

1. Déchiffrer du morse avec un arbre binaire

Rappels sur le morse :

Le code morse est composé de deux symboles, **le point (.)** et l**e tiret (-)**. Le séparateur entre les codes de deux lettres est un espace et le séparateur entre deux mots est le symbole / .

Exemple : `BONJOUR MONSIEUR` va s'écrire en morse :

```
-... --- -. .--- --- ..- .-./-- --- -. ... .. . ..- .-.
```

Ce code peut-être utiliser sous de nombreuses formes. Les signes graphiques (point ou tiret) peuvent être traduit en impulsions électriques, sonores ou lumineuses.

Conventions :

- Le **point** est traduit par une **impulsion brève**.
- Le **tiret** est traduit par une **impulsion longue** dont la durée est égale à trois impulsions brèves.
- L'espace entre deux signaux (point ou tiret) se traduit par un **silence court** dont la durée est égale à une impulsion brève.
- L'**espace** qui sépare le code morse de deux lettres se traduit par un **silence moyen** dont la durée est égale à trois impulsions brèves.
- Le **slash** qui sépare le code morse de deux mots se traduit par un **silence long** dont la durée est égale à sept impulsions brèves.

Code morse pour les lettres majuscules :

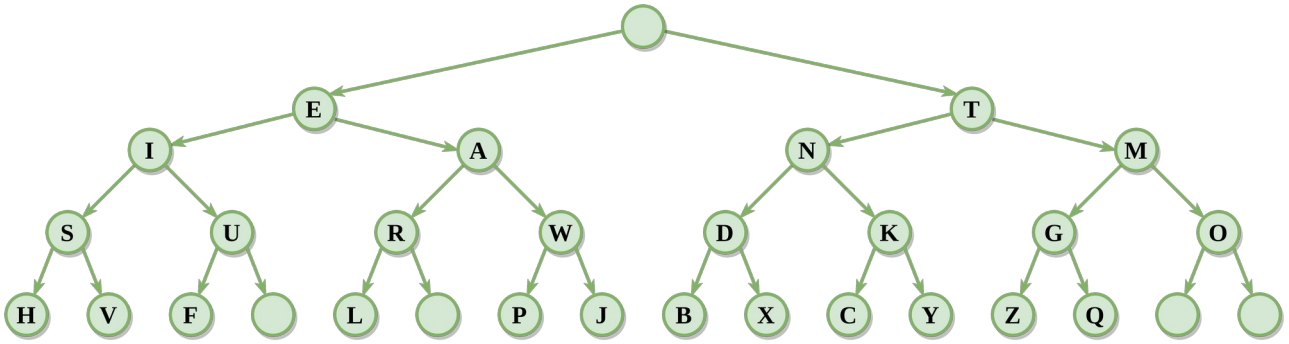
A .-	J .---	S ...
B -...	K -.-	T -
C -.-.	L .-..	U ..-
D -..	M --	V ...-
E .	N -.	W .--
F ..-.	O ---	X -..-
G --.	P .--.	Y -.--
H	Q --.-	Z --..
I ..	R .-.	

Étape 1 : Construire l'arbre binaire parfait pour déchiffrer le morse

Pour déchiffrer le morse, on peut construire un arbre binaire parfait dont les clés sont les caractères en majuscule de l'alphabet ou l'espace.
Pour connaître la position d'un caractère dans l'arbre connaissant son code morse, on utilise le principe suivant :

- On part de la racine de l'arbre ;
- Si dans le code morse on rencontre un point : "." on passe au sous-arbre gauche ;
- Si dans le code morse on rencontre un tiret: "-" on passe au sous-arbre droit ;
- Si on rencontre un séparateur : espace, slash ou fin de code : on récupère la clé dans l'arbre c'est à dire le caractère en majuscule.

Arbre binaire pour les codes morse :



Exemple : A l'aide de l'arbre ci-dessus déchiffrer le message en morse suivant :

```
-... --- -. -. ./.--- --- ..- .-. -. . .
```

BONNE JOURNEE

Fonction **parfait( )** pour créer l'arbre binaire : L'objectif est de créer l'arbre parfait ci-dessus, à partir de son parcours en profondeur préfixe (ngd).
Donner ci-dessous, le parcours préfixe (NGD) de l'arbre ci-dessus sous forme d'une chaîne de caractères, en mettant un espace pour les nœuds sans caractère :

" EISHVUF ARL WPJTNDBXKCYMGZQO "

Pseudo code de la fonction parfait( ) :

```
Fonction parfait(ngd) :
    clé = 1er_caractère_de_ngd
    AB1 = arbre_binaire(clé)
    Si longueur(ngd) > 1 :
        ngd_g = 1ère_moitié_de_ngd_privé_du_1er_caractère
        AB1.sag = parfait(ngd_g)
        ngd_d = 2ème_moitié_de_ngd_privé_du_1er_caractère
        AB1.sad = parfait(ngd_d)
    retourner AB1
```

• Récupérer le fichier **morse.py** en ressource.
• Compléter dans ce fichier le code python de la fonction **parfait(ngd)**.
• Appliquer la fonction **parfait(ngd)** à la chaîne correspondant au parcours préfixe de l'arbre parfait pour les codes morse obtenue ci-dessus .
• Vérifier dans la console avec la fonction **print( )** que la fonction fonctionne correctement.

Étape 2 : Déchiffrer du morse à l'aide de l'arbre parfait

Pour déchiffrer une chaîne de caractères codée en morse, on va rédiger le code de la fonction **dechiffrer(chaine_morse)** en utilisant comme auxiliaire l'arbre parfait créé à l'étape 1.

Principe : On initialise une chaîne_retour à vide. On part de la racine de l'arbre parfait. On commence à parcourir les caractères de la chaîne codée en morse en appliquant les règles suivantes :

- Si le caractère est un point : "." on passe au sous-arbre gauche ;
- Si le caractère est un tiret: "-" on passe au sous-arbre droit ;
- Si le caractère est un séparateur : espace, slash ou fin de chaîne : on récupère la clé dans l'arbre parfait c'est à dire le caractère en majuscule et on ajoute cette clé à la chaîne_retour, on ajoute éventuellement un espace dans le cas du slash. Puis on se replace à la racine de l'arbre.

• Rédiger dans le fichier **morse.py** le code python de la fonction **dechiffrer(chaine_morse)**.
• Ne pas oublier de commenter le code et de renseigner le docstring de la fonction.
• Prévoir une série de tests personnalisés pour vérifier que la fonction **dechiffrer( )** fonctionne correctement.

Il faudra rendre sur pronote un dossier zippé dont le nom sera au format **morse_nom_prenom.zip** .

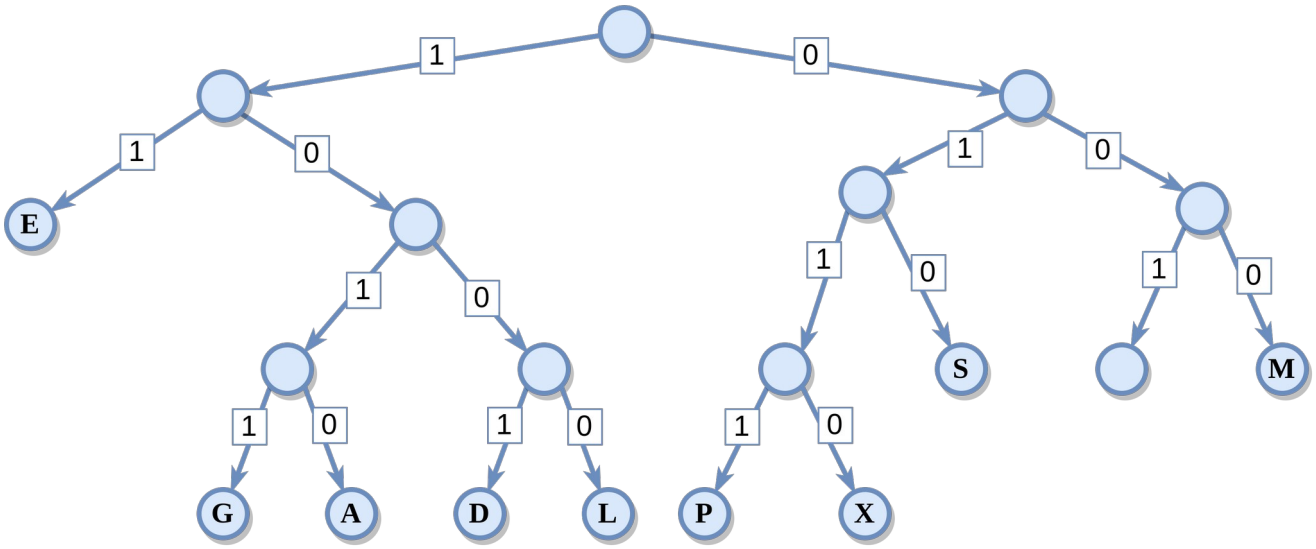
1. Déchiffrer du code de Huffman

Étape 3 : Déchiffrer du code de Huffman

Principe : Le principe du codage de Huffman est de compresser un texte en remplaçant les codes ASCII des caractères par des codes binaires plus petit pour les caractères les plus fréquents.
Il faut d'abord générer un arbre de Huffman à partir du calcul des fréquences de chaque caractère dans le texte. Dans cet arbre chaque caractère est placé dans une feuille et on retrouve son code binaire en parcourant l'arbre de haut en bas depuis la racine avec la règle suivante :

- Si on choisit le sous-arbre gauche cela code un "1" ;
- Si on choisit le sous-arbre droite cela code un "0" ;

Exemple d'arbre de Huffman



En utilisant cet arbre de Huffman, donner les codes binaires des caractères suivants :

"E" : 11

"L" : 1000

" " : 001

" " :

En utilisant cet arbre de Huffman, déchiffrer le code binaire suivant :

```
1101101100001111000110011001110010001101001010101111
```

EXEMPLE DE MESSAGE

Fonction **dechiffrer(chaine_binaire)** :
L'objectif est de programmer en python la fonction qui permet de déchiffrer le code binaire de Huffman en utilisant l'arbre de Huffman comme auxiliaire.

• Compléter le pseudo code suivant :

```
Fonction dechiffrer(chaine_binaire) :
    chaine_retour = ""
    AB1 = arbre_huffman
    Pour chaque caractère  c  de la chaine_binaire :
        Si c = '1' :
            AB1 = AB1.sag
            Si on est sur une feuille :
                chaine_retour += clé_feuille
                AB1 = arbre_huffman
        Sinon :
            AB1 = AB1.sad
            Si on est sur une feuille:
                chaine_retour += clé_feuille
                AB1 = arbre_huffman
    retourner chaine_retour
```

• Récupérer le fichier **huffman.py** en ressource.
• Rédiger dans le fichier **huffman.py** le code python de la fonction **dechiffrer(chaine_binaire)**.
• Ne pas oublier de commenter le code et de renseigner le docstring de la fonction.
• Vérifier que cela fonctionne correctement en appliquant cette fonction à la chaîne binaire donnée précédemment.

Étape 4 * : Arbre et code de Huffman

• Rédiger le code d'une fonction **frequence(chaine)** qui retourne un dictionnaire dont les clés sont les caractères contenus dans la chaîne passée en paramètre et les valeurs sont les fréquences d'apparition de ces caractères dans cette chaîne.
• Rédiger le code d'une fonction **arbre_huffman(dico)** qui retourne l'arbre binaire de Huffman obtenu à partir du dictionnaire des fréquences passé en paramètre. Faire des recherche sur internet pour comprendre la construction de cet arbre.
• Rédiger le code d'une fonction **code_huffman(chaine, arbre)** qui retourne la chaîne binaire correspondant à la chaîne de caractères passée en paramètre codée à l'aide du code de Huffman : chaque caractère est remplacé par son code obtenu par le parcours dans l'arbre de Huffman.
• Vérifier que tout fonctionne en utilisant notamment la fonction **dechiffrer( )** de l'étape 3.