REBER Elise 21929616
Ahmed Iacob Essallami Mana 21926101

Projet Huffman

1. Conception des arbres

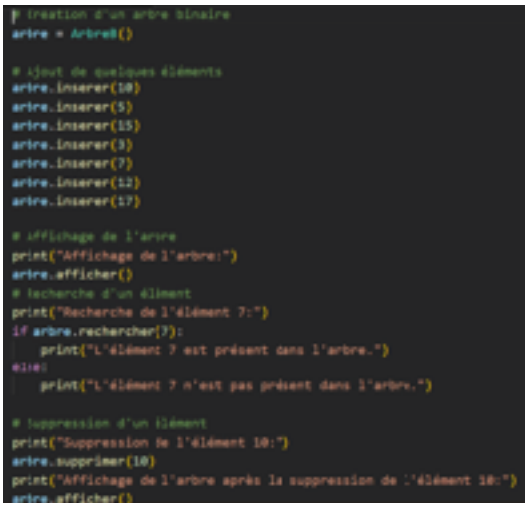
Pour créer un arbre binaire, nous avons utilisé deux classes : la classe **Sommet** et la classe **ArbreB**. Chaque élément de la classe **Sommet** correspond à un sommet de l'arbre et permet d'avoir son étiquette. Chaque élément de la classe **ArbreB** correspond à un arbre binaire composé de sommets de la classe **Sommet**.

La classe **ArbreB** permet de construire un arbre binaire sur le modèle de l'arbre de recherche binaire : pour chaque sommet inséré, la méthode **ajouter_sommet** vérifie si la racin est nulle, auquel cas le nouveau sommet devient la racine. Si la racine n'est pas vide, alors la méthode appelle la méthode **_ajouter_sommet** qui compare l'étiquette du nouveau sommet à l'étiquette de la racine : si l'étiquette du nouveau sommet est plus petite que celle de la racine, alors la méthode répète l'opération récursivement sur le sous-arbre gauche et si l'étiquette du nouveau sommet est plus grande que celle de la racine, alors elle répète l'opération récursivement sur le sous-arbre droit.

Pour rechercher un sommet dans l'arbre, la méthode **rechercher** vérifie que la racine ne soit pas vide : si elle l'est, elle renvoie False. Sinon, elle fait appel à la méthode **_rechercher** qui compare l'étiquette du sommet courant (à partir de la racine) à l'étiquette du sommet recherché : si l'étiquette du sommet courant correspond à celle du sommet rechercher, elle renvoie True. Si ce n'est pas le cas et que l'étiquette du sommet courant est plus petite que celle du sommet recherché, la méthode répète récursivement l'opération sur le sous-arbre droit. Sinon, elle répète récursivement l'opération sur le sous-arbre gauche jusqu'à ce que le sommet soit trouvé ou que l'arbre soit entièrement parcouru.

La classe **ArbreB** peut aussi fusionner deux arbres binaires et décomposer un arbre binaire en deux arbres. La méthode **fusionner** fusionne deux arbres en créant un nouvel objet **ArbreB** et en appelant la méthode **_fusionner** qui prend chaque sommet du premier arbre et en additionnant son étiquette et celle du deuxième arbre pour obtenir les sommets du nouvel arbres. La méthode **decomposer_arbres** crée deux nouveaux objets **ArbreB** et appelle la fonction **_decomposer_sommets** qui remplit le premier objet **ArbreB** avec les sommets du sous-arbre gauche et le deuxième objet avec les sommets du sous-arbre droit.

Pour supprimer un sommet, la classe **ArbreB** utilise la méthode **supprimer** qui commence par vérifier si la racine est vide : si c'est le cas, elle renvoie la racine. Sinon, elle recherche le sommet correspondant à la valeur dans l'arbre : si cette valeur ne correspond à la valeur de l'étiquette d'aucun sommet, elle renvoie False. Sinon, elle vérifie si le sommet à supprimer a des enfants. S'il n'a pas d'enfants, elle le supprime. S'il a un seul enfant, elle rattache cet enfant à son parent. S'il a deux enfants, elle utilise la méthode **_trouver_successeur** pour pouvoir trouver le successeur et y rattacher les enfants du sommet à supprimer avant de le supprimer.

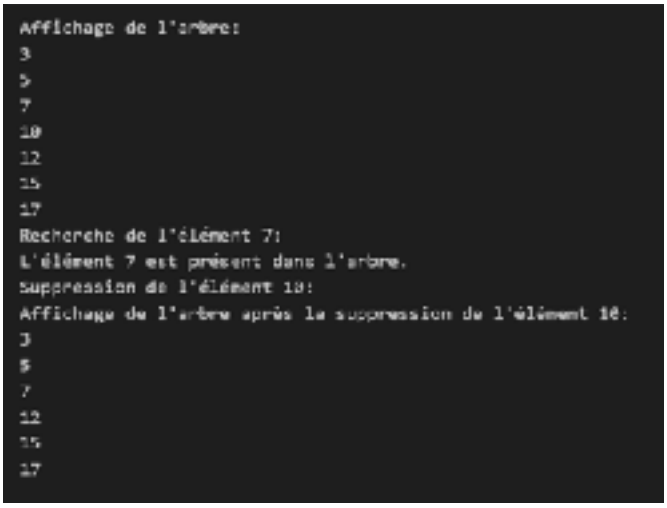


```
# création d'un arbre binaire
arbre = ArbreB()

# ajout de quelques éléments
arbre.inserer(10)
arbre.inserer(5)
arbre.inserer(15)
arbre.inserer(3)
arbre.inserer(7)
arbre.inserer(12)
arbre.inserer(17)

# affichage de l'arbre
print("Affichage de l'arbre:")
arbre.afficher()
# recherche d'un élément
print("Recherche de l'élément 7:")
if arbre.rechercher(7):
    print("L'élément 7 est présent dans l'arbre.")
else:
    print("L'élément 7 n'est pas présent dans l'arbre.")

# suppression d'un élément
print("Suppression de l'élément 10:")
arbre.supprimer(10)
print("Affichage de l'arbre après la suppression de l'élément 10:")
arbre.afficher()
```

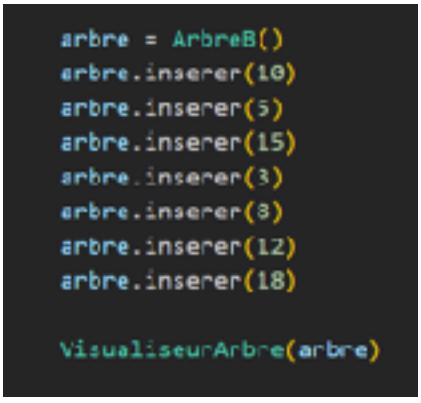


```
Affichage de l'arbre:
3
5
7
10
12
15
17
Recherche de l'élément 7:
L'élément 7 est présent dans l'arbre.
Suppression de l'élément 10:
Affichage de l'arbre après la suppression de l'élément 10:
3
5
7
12
15
17
```

Figure 1 : Exemple d'utilisation de la classe **ArbreB**

Figure 2 : Résultat des instructions précédentes

Pour afficher les arbres, nous avons utilisé la bibliothèque **Tkinter** pour créer une classe **VisualiseurArbre** qui dessine un arbre binaire avec la méthode **dessiner_arbre** : cette méthode prend chaque sommet, dessine un cercle sur un canvas et écrit l'étiquette du sommet dans le cercle. Elle dessine ensuite un trait vers les enfants du sommet si besoin et recommence l'opération.



```
arbre = ArbreB()
arbre.inserer(10)
arbre.inserer(5)
arbre.inserer(15)
arbre.inserer(3)
arbre.inserer(8)
arbre.inserer(12)
arbre.inserer(18)

VisualiseurArbre(arbre)
```

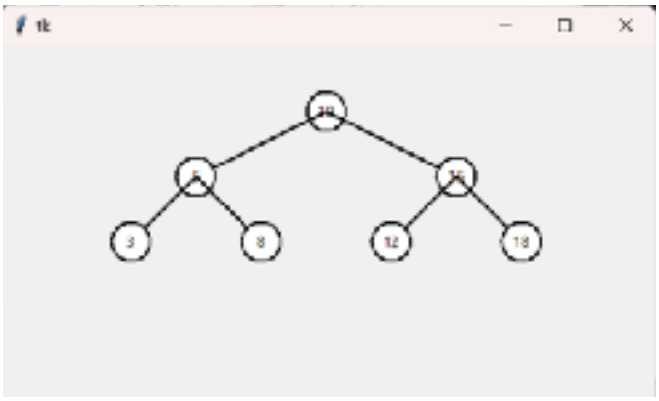


Figure 3 : Exemple de création d'arbre à afficher

Figure 4 : Arbre résultant des instructions précédentes

2. Cryptage de Huffman

On a ensuite créé une classe **ArbreHuffman** héritant de la classe **ArbreB** qui représente chaque nœud de l'arbre binaire de cryptage. Cette classe ajoute à chaque sommet l'attribut **freq** qui représente sa fréquence dans le texte sur lequel se base notre arbre de cryptage.

Avec la fonction **huffman_encoding**, on construit cet arbre qui est un objet **ArbreHuffman**. On commence par calculer la fréquence d'occurrence de chaque lettre et caractère dans le texte en utilisant la méthode **Counter** de la bibliothèque **collections**. Elle enregistre ces fréquences dans un dictionnaire **poucentage_occurrences**, puis crée une liste d'objets **ArbreHuffman** composés d'un unique sommet dont l'étiquette est la lettre et dont la fréquence est la fréquence d'occurrence dans le texte témoin.

Elle parcourt ensuite la liste d'objets pour les fusionner et créer de nouveaux objets **ArbreHuffman** en additionnant les fréquences pour obtenir la fréquence de la nouvelle racine et effectue cette opération jusqu'à ce qu'il ne reste plus d'un seul objet **ArbreHuffman** dans la liste, selon l'algorithme donné, qui est notre arbre binaire de cryptage basé sur notre texte témoin.

Elle parcourt ensuite l'arbre binaire de cryptage pour y ajouter le code binaire de chacun de ses sommets.

On peut ensuite afficher cet arbre dans le terminal avec la fonction **afficher_arbre** :

Figure 5 : affichage dans un éditeur de texte de l'arbre binaire de cryptage basé sur le texte « portez ce vieux whisky au juge blond qui fume '. »

Pour coder un texte selon cet arbre, on crée un dictionnaire de correspondance entre chaque lettre et son code binaire avec la fonction **construire_dictionnaire_encodage** qui parcourt l'arbre récursivement, puis on utilise la fonction **encoder_texte** pour faire correspondre chaque lettre du texte à son code binaire.

On a ensuite créé une interface graphique permettant de coder un texte selon cet arbre de codage en utilisant **Tkinter**. On a utilisé une fonction **cliquer_pour_encoder** qui permet de crypter un texte entré dans la zone d'entrée en appuyant sur le bouton encoder. On a également affiché l'arbre binaire de cryptage, avec un code inspiré de la classe **VisualiseurArbre**, sur lequel on peut se déplacer sur l'interface graphique avec la fonction **draw_tree**.

Figure 6 : Ici, on entre le texte à encoder « bonjour » et on obtient le code binaire correspondant.

On voit aussi une partie de l'arbre binaire de cryptage sur la gauche.

On peut rentrer le texte à encrypter et la fenêtre nous renvoie le texte en binaire.

3. Décryptage

Afin de décoder un texte codé selon notre arbre binaire de cryptage, on utilise la fonction **decoder_texte** qui recherche pour chaque bit s'il s'agit d'une feuille ou non : si oui, il renvoie la lettre correspondant à la feuille. Si ce n'est pas le cas et le bit est égal à 0, il cherche dans le sous-arbre gauche du nœud courant. Si ce n'est pas le cas et le bit est égal à 1, il cherche dans le sous-arbre droit du nœud courant. Il renvoie le texte décodé.

On a créé une interface graphique avec la bibliothèque **Tkinter** pour pouvoir entrer un texte encrypté et récupérer le texte décrypté en appuyant sur le bouton **décoder** grâce à la fonction **cliquer_pour_décoder**.

Figure 7 : Ici, on entre le code binaire « 110111100111110011 1010110011011101001 » et on obtient le code décrypté « bonjour ».