Exercice 1 (6 points)

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.

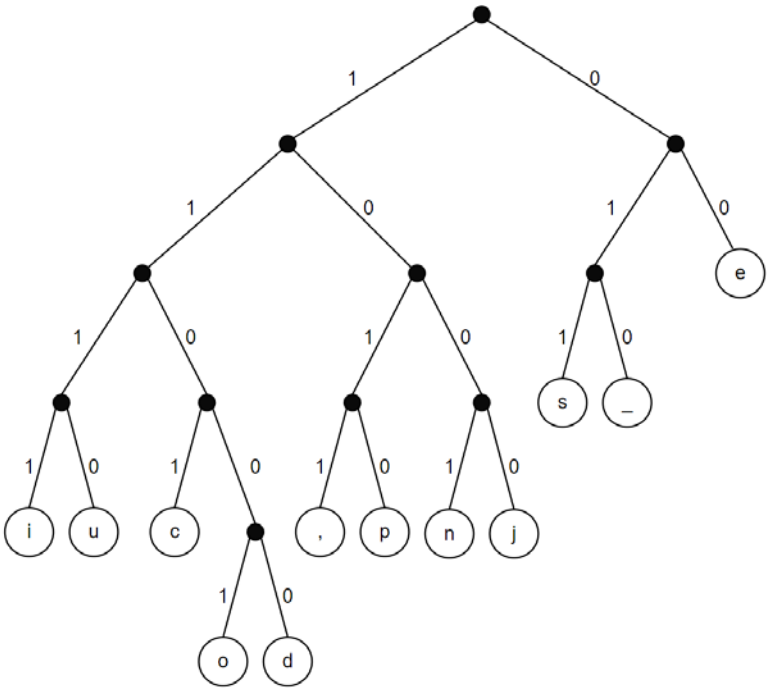

Figure 1. Exemple d'arbre de codage

1. Écrire le mot binaire qui sera utilisé pour encoder le caractère espace, représenté par le symbole _ dans l'arbre.

2. Déterminer le texte codé par le mot binaire `000111010111111001100 1`.

3. Citer le type de parcours de l'arbre qui permettrait d'obtenir les symboles classés par taille d'encodage croissante.

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.

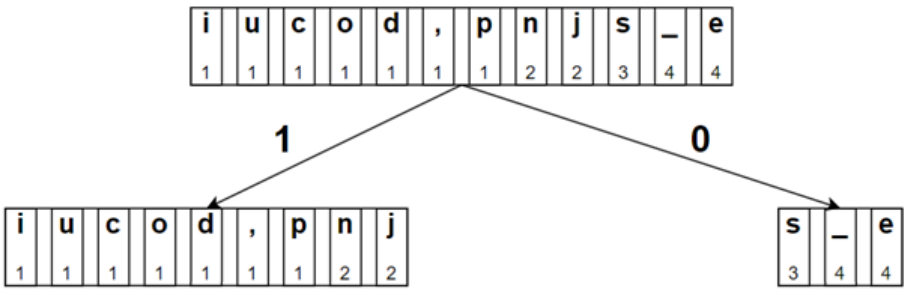

Figure 2. Le résultat de l'étape 2

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

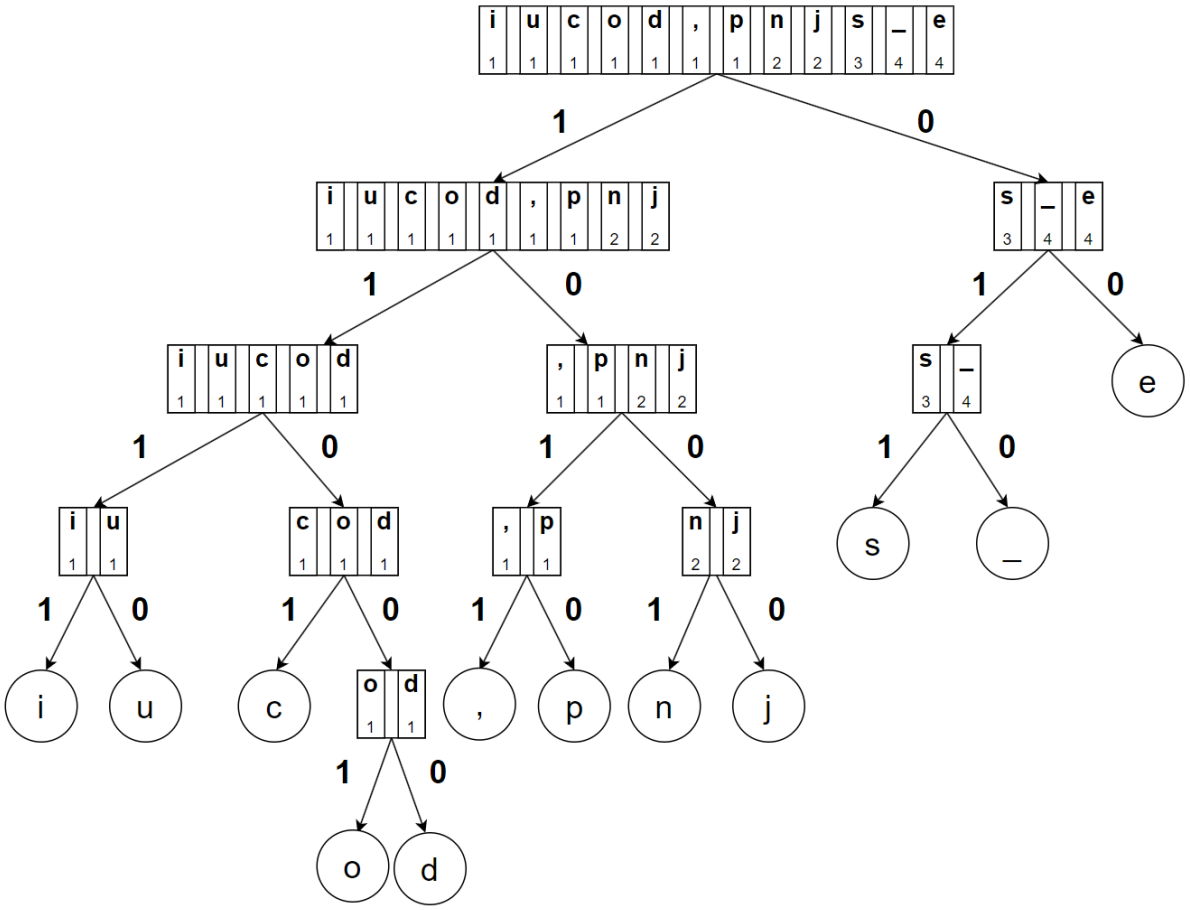

Figure 3. Arbre de codage obtenu par l'algorithme de Shannon-Fano

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.

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`

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. Recopier et compléter les lignes 8 et 10 du code de la fonction `creer_dico_occ`.

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.

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. Recopier et 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" + ...
```

11. Décrire ce qui garantit la terminaison de la fonction récursive `shannon`.

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
```