Première	ACTIVITÉ	NSI
	Codage de Huffman	

A. Introduction : Les marmottes au sommeil léger

➢ *Dans cette activité, vous allez aider des marmottes à construire leur terrier. Parviendrez-vous à faire en sorte que le terrier convienne à l'ensemble des marmottes ? Parviendrez-vous à trouver la meilleure stratégie ?*

➢ Pour jouer, vous vous placez par groupe de 4.

 1ère étape : *Le contexte.*

Un groupe de marmottes, moyennement satisfaites de leur terrier actuel, décide de concevoir un nouveau terrier et de le creuser avant l'hiver. Pour ce faire les marmottes doivent respecter trois règles :

- Règle 1 : À partir de l'entrée on peut construire deux couloirs, et au bout de chaque couloir on peut faire un embranchement vers deux autres couloirs, mais pas plus (au risque de faire s'écrouler l'édifice).
- Règle 2 : Les marmottes vont chacune occuper une salle différente (pour ne pas se réveiller les unes les autres) et forcément une salle qui est tout au bout d'un couloir. Pour des marmottes au sommeil léger il n'est pas envisageable de dormir dans une salle à un embranchement, car les marmottes qui seraient au-delà de cet embranchement leur marcheraient dessus en entrant/sortant, et cela ruinerait leur hibernation.
- Règle 3 : Chaque marmotte se réveille un nombre précis de fois dans l'hiver.

 2ème étape : *Distribution du matériel et objectif du terrier.*

- Pour les couloirs : vous pouvez les assembler entre eux en respectant la règle n°1.
- Pour les marmottes : le nombre écrit sur chacune est le nombre de fois où elle se réveille dans l'hiver.

Marmottes	**A**licia	**B**ernardo	**E**liane	**I**sidore	**L**ouison	**R**oberta	**S**uzanna	" "
Réveils	6	5	4	2	2	5	2	4

On compte les déplacements des marmottes de la façon suivante. Une marmotte dormant à 4 couloirs de l'entrée se réveillant 5 fois dans l'hiver va parcourir 4x5=20 couloirs aller et retour (pour simplifier on ne va compter que les allers). On fait la somme des déplacements de toutes les marmottes.

Comme les pas de marmottes émettent de légères vibrations et que nos marmottes ont vraiment le sommeil léger, on veut rendre le plus petit possible l'ensemble des déplacements du groupe.

L'objectif est donc de faire la somme des déplacements de toutes les marmottes en rendant ce nombre le plus petit possible.

 3ème étape : *Et maintenant, creusez !*

- Avec vos camarades, essayez d'assembler les couloirs entre eux et de mettre une

marmotte à chaque bout. Comptez ensuite le nombre total de déplacements, puis remplissez le tableau au verso pour le 1er essai :

Numéro de l'essai	n°1	n°2	n°3	n°4	n°5	n°6
Calculs effectués						
Nombre total de déplacements						

- Faites un nouvel essai en essayant de rendre plus petit le nombre total de déplacements. Faites plusieurs essais, puis appelez le professeur.
- Pour la meilleure solution : Comment les marmottes sont-elles placées ? Faire un schéma :

- Quelles différentes stratégies avez-vous mises en place ?

- Quelle stratégie a permis que le nombre total de déplacements soit le plus petit possible ?

Les marmottes feraient-elles de l'informatique ?
Rien à voir… ou tout à voir ?

- Compléter le tableau des effectifs des caractères composant la phrase “BARBARA RASE BASILE LE BARBIER” (espace compris).

Lettre	A	B	E	I	L	R	S	“ ”
Effectif								

- Quelle analogie peut-on faire avec la colonie de marmottes ?

- Retourner le terrier réalisé. Sur chaque branche gauche indiquer le chiffre 1 et sur chaque branche droite le chiffre 0.
À partir de la sortie du terrier, en suivant le chemin le plus rapide, établir un séquence de 0 et 1 pour accéder à chacune des lettres.

Lettre	A	B	E	I	L	R	S	“ ”
Code								

➢ *Cette association s’appelle un dictionnaire. Il est appelé code de Huffman du nom du chercheur en théorie de l'information qui l’a inventé*.

- À l’aide de ce dictionnaire, écrire la phrase d’origine à l’aide d’une seule séquence de 0 et de 1.

- Combien de bits a-t-on utilisés ? Comparer ce résultat avec le nombre de bits nécessaires pour encoder ce même texte en ASCII (8 bits) ? Que cela permet-il de faire ?

- Ce code est-il facilement décodable ? Pourquoi ?

➢ *On dit que ce code est préfixe.*

B. De la théorie

Le codage de Huffman (1952) est une méthode de compression statistique de données qui permet de réduire la longueur du codage d'un alphabet. C'est un code de longueur variable optimal, c'est-à-dire que la longueur moyenne d'un texte codé est minimale. On observe des réductions de taille de l'ordre de 20 à 90%.

B.1. Le principe

➢ Le principe de l'algorithme de Huffman consiste à recoder les octets rencontrés dans un ensemble de données source avec des valeurs de longueur binaire variable.

L'unité de traitement est ramenée au bit. Huffman propose de recoder les données qui ont une occurrence très faible sur une longueur binaire supérieure à la moyenne, et recoder les données très fréquentes sur une longueur binaire très courte. Ainsi, pour les données rares, nous perdons quelques bits regagnés pour les données répétitives.

 Par exemple, dans un fichier ASCII le « w » apparaissant 10 fois aura un code très long : 010100000100. Ici la perte est de 40 bits (10 x 4 bits), car sans compression, il serait codé sur 8 bits au lieu de 12.

Par contre, le caractère le plus fréquent comme le « e » avec 200 apparitions sera codé par 1. Le gain sera de 1400 bits (7 x 200 bits).

On comprend l'intérêt d'une telle méthode.

➢ De plus, le codage de Huffman a une propriété de préfixe : une séquence binaire ne peut jamais être à la fois représentative d'un élément codé et constituer le début du code d'un autre élément.

Si un caractère est représenté par la combinaison binaire 100 alors la combinaison 10001 ne peut être le code d'aucune autre information. Dans ce cas, l'algorithme de décodage interpréterait les 5 bits comme deux mots : 100-01. Cette caractéristique du codage de Huffman permet une codification à l'aide d'une structure d'arbre binaire.

B.2. La méthode

Dans l'exemple qui suit on cherche à obtenir un codage de Huffman pour le mot-phrase « ABRACADABRA »

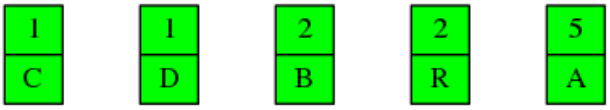
➢ Préalable

Tout d'abord on effectue, un relevé des effectifs (ou fréquences) d'apparition de chacune des lettres (ou plus généralement caractères) qui constituent la phrase.

Lettre	**A**	**B**	**R**	**C**	**D**
Effectif	**5**	**2**	**2**	**1**	**1**

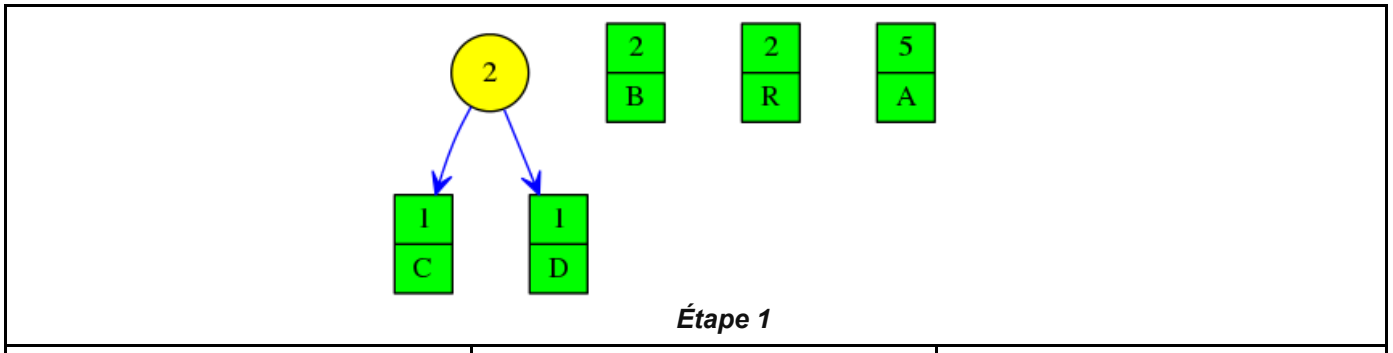
➢ L'algorithme

 Au départ, chaque lettre est associée à une feuille d'étiquette correspondant à son effectif (ou sa fréquence) d'apparition. En considérant une feuille comme un arbre à un nœud, il y a autant d'arbres que de lettres.

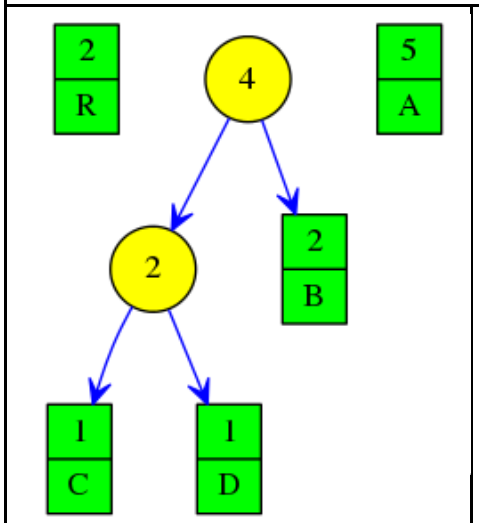


Étape 0

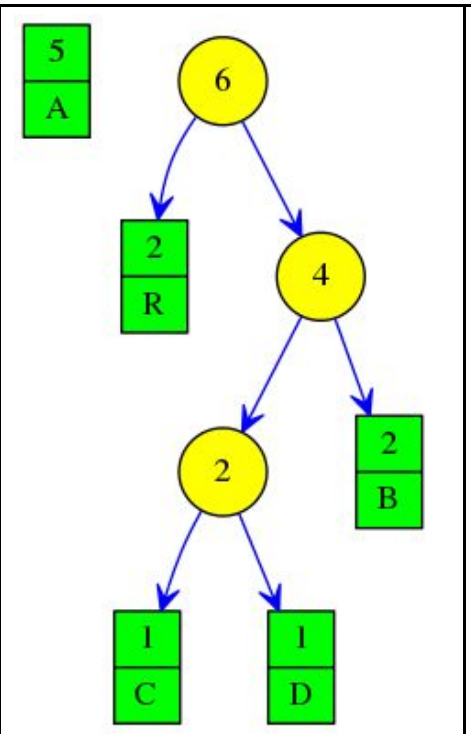
 L'algorithme choisit à chaque étape les deux arbres d'étiquettes minimales, soit x et y, et les remplace par l'arbre formé de x et y et ayant comme étiquette la somme de l'étiquette de x et de l'étiquette de y. La figure ci-après représente les étapes de la construction d'un code de Huffman pour l'alphabet source {A, B, R, C, D}, avec les effectifs eff(A)=5, eff(B)=2, eff(R)=2, eff(C)=1 et eff(D)=1.



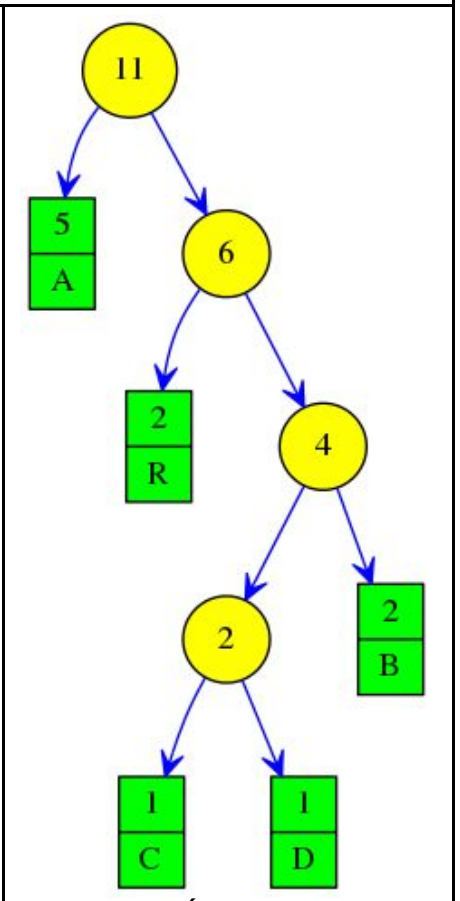
Étape 1



Étape 2



Étape 3



Étape 4

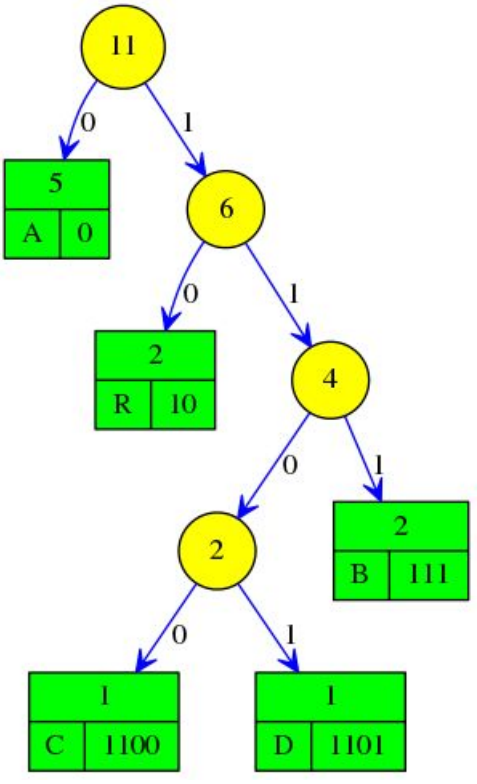
- Le code d'une lettre est alors déterminé en suivant le chemin depuis la racine de l'arbre jusqu'à la feuille associée à cette lettre en concaténant successivement un 0 ou un 1 selon que l'arc suivi est à gauche ou à droite.

Ainsi, grâce à l'arbre ci-contre on obtient les traductions,

A :0, B :111, R :10, C :1100, D :1101.

Étape Code de Huffman

Ce « dictionnaire » constitue ce qu'on appelle le **code de Huffman** du codage.



- « ABRACADABRA » est alors codé par

0 1 1 1 1 0 0 1 1 0 0 0 1 1 0 1 0 1 1 1 1 0 0

Étape Codage de Huffman

Pour décoder, on lit simplement la chaîne de bits de gauche à droite. Le seul découpage possible, grâce à la propriété du préfixe, est 0-111-10-0-1100-0-1101-0-111-10-0.

➢ Synthèse

- Le code ASCII de « ABRACADABRA » occupe 11x8 = 88 bits alors que son codage de Huffman occupe 23 bits !
  Cependant pour décoder, il faut transmettre le code de Huffman associé : 5 lettres x 8 bits et 14 bits pour les codes associés, soit un total de 54 bits supplémentaires (non négligeable ici, dans le cadre de notre courte chaîne !).
  Le taux de compression se calcule alors par :

$$t_{comp} = 1 - \frac{V_{final}}{V_{initial}} = 1 - \frac{23+54}{88} = 12,5\%$$

où les volumes de la donnée sont exprimés en nombre de bits (ou d'octets)

- Il n'y a pas un unique code de Huffman possible. Par exemple pour l'étape 2, on aurait pu associer les feuilles d'étiquette 2 attachées aux lettres B et R plutôt que le nœud d'étiquette 2 et la feuille attachée à B. De plus, sur les arcs issus d'un même nœud on peut intervertir les étiquettes 0 et 1. On montre mathématiquement que quelque soit le code de Huffman utilisé, le codage garde la même taille.

B.3. Exercices

➢ ***Exercice 1 :***

1) Construisez un codage de Huffman de la séquence d'ADN suivante : « CAGATAAGAGAA ».

2) Calculez le taux de compression qu'il permet de réaliser (en tenant compte du dictionnaire).

➢ ***Exercice 2*** (à la maison pour les plus courageux) ***:***

1) Utilisez le tableau ci-dessous pour déterminer un codage de Huffman de la langue française.

Lettre	Fréquence	Lettre	Fréquence
A	8.40 %	N	7.13 %
B	1.06 %	O	5.26 %
C	3.03 %	P	3.01 %
D	4.18 %	Q	0.99 %
E	17.26 %	R	6.55 %
F	1.12 %	S	8.08 %
G	1.27 %	T	7.07 %
H	0.92 %	U	5.74 %
I	7.34 %	V	1.32 %
J	0.31 %	W	0.04 %
K	0.05 %	X	0.45 %
L	6.01 %	Y	0.30 %
M	2.96 %	Z	0.12 %

2) Calculer le taux de compression moyen d'un texte en français par un codage de Huffman (on ne considérera que des textes en majuscules et encodés en ASCII ; on ne tiendra pas compte du dictionnaire).

C. À la pratique

L'objectif est ici d'écrire un programme en Python 3 qui effectue un codage de Huffman pour une chaîne donnée par l'utilisateur.

C.1. La modélisation

C.1.1. Les structures de données

A partir des types de variables standards de Python, on va construire les structures de données des objets à manipuler :

➢ ***message** d'entrée ou sortie :* variable de type ……..

➢ ***code** de Huffman* : variable de type ……..

 Dans le cadre de notre exemple, modéliser le code de Huffman trouvé :

➢ ***feuille, arbre et forêt***

Lorsque l'on parcourt « naturellement » un arbre binaire on regarde tout d'abord l'étiquette de son nœud racine, puis on regarde un de ses deux nœuds fils que l'on considère lui-même comme nœud racine d'un sous-arbre. Partant de cette constatation, on peut modéliser un arbre binaire A par une variable de type liste de longueur 3 comme suit :

$A = [r, A'_g, A'_d]$ où r est l'étiquette de la racine, A'_g le sous-arbre gauche et A'_d le sous-arbre droit. Cette structure est dite récursive car elle est définie à partir de constituants qui sont eux-même des structures de même type.

Or une feuille a est un arbre sans sous-arbre (donc de hauteur 0), on aurait pu alors la modéliser par une liste à un élément, son étiquette. Cependant, étant donné qu'on a besoin ici de la lettre qui lui est rattachée on optera plutôt pour :

$a = [r, s]$ où r est l'étiquette de la feuille et s sa sous-étiquette (la lettre).

Ainsi une feuille se différencie d'un arbre de hauteur non nulle par le fait qu'elle est une liste de longueur 2 et non 3.

arbre non feuille : variable de type *list* à 3 éléments **feuille** : variable de type *list* à 2 éléments

 Dans le cadre de notre exemple, modéliser l'arbre de Huffman trouvé :

Une forêt F est une collection d'arbre donc on la modélisera par une liste : $F = [A_1, A_2, ..., A_n]$

forêt : variable de type *list*

C.1.2. Les fonctions

A partir des structures de données choisies, on va maintenant décomposer la tâche globale à réaliser en des sous-tâches effectuées par des fonctions. Pour cela, nous allons suivre scrupuleusement le déroulement de l'algorithme, et associer une fonction à chaque étape :

➢ Étape 0

Nom de la fonction	***string2Forest***
Structure du paramètre d'entrée	
Structure de la valeur de retour	
Rôle	
Fonction(s) ou méthode(s) utile(s)	**maListe.*append*(e)** : méthode qui ajoute l'élément e à la liste maliste ; équivalent à **maliste+=[e]** **maChaine.*count*(car)** : méthode qui compte les occurrences du caractère car dans le texte maChaine. **maChaine.*replace*(old,new)** : fonction qui renvoie une chaîne où les occurrences de old on été remplacées par new.

➢ Étape *n* à Étape *n+1*

Nom de la fonction	***huffmanStep***
Structure du paramètre d'entrée	
Structure de la valeur de retour	
Rôle	
Fonction(s) ou méthode(s) utile(s)	

➢ Étape 1 à dernière Étape numérotée

Nom de la fonction	***forest2Htree***
Structure du paramètre d'entrée	
Structure de la valeur de retour	
Rôle	
Fonction(s) ou méthode(s) utile(s)	**maListeDeListes.*sort(key=lambda x:x[i])*** : méthode qui trie une liste de liste par ordre croissant des valeurs au rang i.

➢ Étape Code de Huffman

Nom de la fonction	***tree2Code***
Structure du paramètre d'entrée	
Structure de la valeur de retour	
Rôle	
Fonction(s) ou méthode(s) utile(s)	

➢ Étape Codage de Huffman

Nom de la fonction	***coding***
Structure du paramètre d'entrée	
Structure de la valeur de retour	
Rôle	
Fonction(s) ou méthode(s) utile(s)	

C.2. L'implémentation

À vous maintenant de coder en Python 3 *string2Forest*, *huffmanStep* et *forest2Htree* !

BON COURAGE !