Université de Caen Normandie, UFR des Sciences
Licence 3

Module " Algorithmique, structures informatiques et cryptologie"
Codage de Huffman ; durée 2h30

Nous avons vu dans le cours le principe du codage d'une source sans mémoire X sur un alphabet $\mathcal{A} = \{a_1, a_2, \ldots, a_K\}$. Rappelons que l'entropie de X est donnée par

$$H(X) = -\sum_{a \in \mathcal{A}} \mathbf{P}(a) \log_2 \mathbf{P}(a),$$

avec $\mathbf{P}(a)$ la probabilité du symbole a. Si ϕ est un code associé à X ($\phi : \mathcal{A} \mapsto \{0,1\}^\star$), alors la longueur moyenne du code est donnée par

$$\ell(\phi) = \sum_{a \in \mathcal{A}} \mathbf{P}(a) \ell(\phi(a)),$$

avec $\ell(x)$ le nombre de bits de x. Le premier théorème de Shannon indique que pour toute source X, il existe un code ϕ tel que

$$H(X) \leq \ell(\phi) < H(X) + 1.$$

L'objectif du TP est d'implémenter en python le codage de Huffman d'un fichier (les fichiers `test.txt` ou `Huffman.txt` ou `exemple.txt` dans notre cas). Rappelons que le code de Huffman est un code préfixe qui vérifie le premier théorème de Shannon. Le programme évaluera l'entropie binaire du fichier source, le taux de compression obtenu, ainsi que la longueur moyenne du code. Le fichier source `tp_huffman.py` à compléter et les fichiers textes sont sur la plateforme ecampus

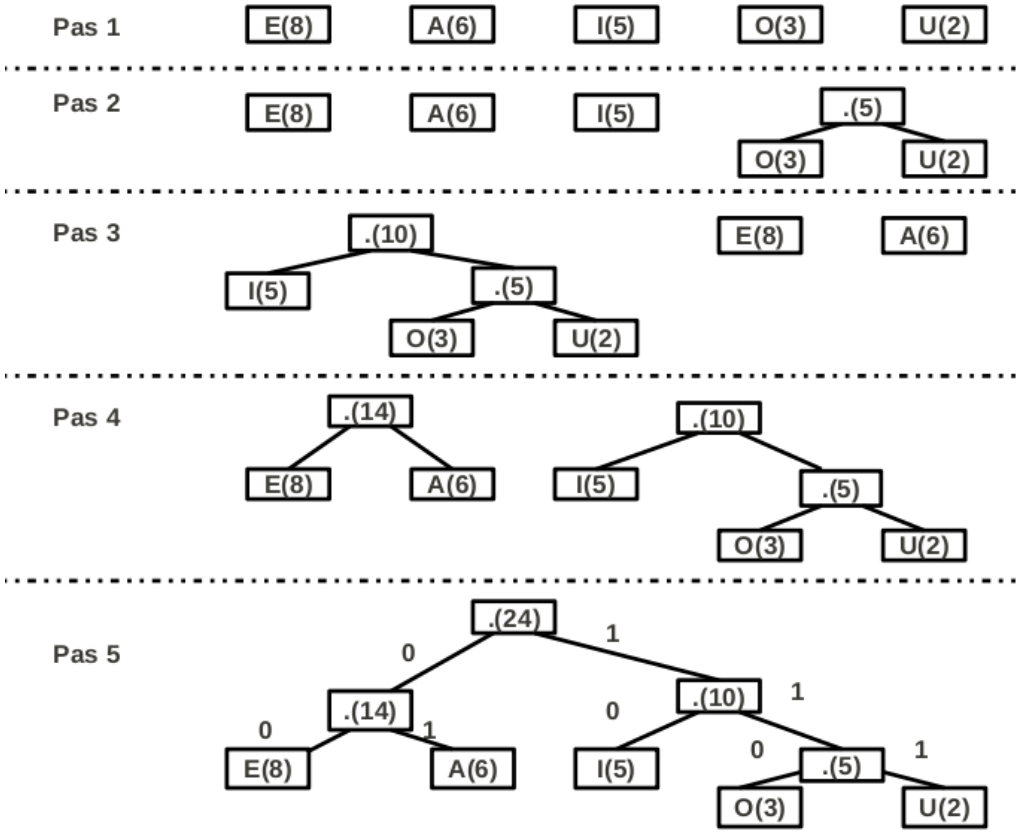
1 Rappels sur le code de Huffman

Le code de Huffman est une technique de compression, sans perte, permettant de coder les lettres et les symboles d'un texte en fonction de leur nombre d'occurrences dans le texte, afin de proposer un système de codage plus court que le code ASCII standard. Pour cela, on construit un arbre de Huffman de la manière suivante (voir exemple ci-dessous) :

1. Les feuilles (nœuds à l'extrémité) de l'arbre contiennent chaque lettre (ou symbole) avec un poids associé correspondant à leur nombre d'occurrences, triées de gauche à droite par ordre décroissant.
2. L'arbre est ensuite créé progressivement en groupant chaque couple de nœuds ayant le poids le plus faible pour donner naissance à une nouveau nœuds père dont le poids est la somme des deux poids des fils.
3. L'arbre est ainsi construit jusqu'à obtenir un seul nœuds qui sera la racine de l'arbre.

Le codage de Huffman consiste ensuite à associer un 0 à chaque branche de gauche et un 1 à chaque branche de droite. Le codage de chaque lettre (ou symbole) se lit alors en partant de la racine jusqu'à arriver à la lettre ou le symbole en question.

Exemple : On considère les 5 voyelles de l'alphabet A (6), E (8), I (5), O (3), U (2) où le chiffre entre parenthèses est leur nombre d'occurrences. L'arbre de Huffman obtenu en appliquant l'algorithme précédent est donné par la figure ci-dessous. A partir de l'arbre, le codage de Huffman obtenu est :A $\mapsto$ 01, E $\mapsto$ 00, I $\mapsto$ 10, O $\mapsto$ 110 et U $\mapsto$ 111. Remarquez que les lettres les plus fréquentes sont codées avec le moins de bits possible.



Construction de l'arbre de Huffman pas à pas

2 Présentation du code source

Afin de faciliter l'implémentation du TP, une partie du code a déjà été développée.
La variable globale `nomfic` est le fichier qui sera lu puis compressé à l'aide du codage de Huffman.
La classe `Noeud` représente les nœuds de l'arbre du codage de Huffman. Un objet de la classe `Noeud` contient les champs suivants :

- `lettre` : si le nœud est une feuille, la lettre est un symbole du texte. Dans le cas contraire, la lettre sera par défaut un point (`'.'`).
- `fils_gauche, fils_droite` : les champs `fils_gauche` et `fils_droite` représentent les sous-arbres gauche et droite du nœud. Par défaut, ceux-ci sont initialisés à `None` (l'arbre vide).
- `nb_occur` : représente le nombre d'occurrences du nœud (nombre entre parenthèses dans l'exemple). Par défaut, cet entier est initialisé à 0.
- `freq` : représente la fréquence du symbole donnée par `nb_occur`/nombre total de symboles du texte. Par défaut, ce flottant est initialisé à 0.0.

Pour créer un objet de la classe `Noeud`, il suffit d'utiliser le constructeur de la classe. Par exemple : `n1=Noeud()` crée un noeud `n1` prenant les valeurs par défaut. Comme aucun paramètre n'est passé au constructeur, les champs prennent les valeurs par défaut. Cette instruction est équivalente à (`n1=Noeud('.',None,None,0,0.0)`) ou (`n1=Noeud(lettre='.',fils_gauche=None,fils_droite=None, nb_occur=0,freq=0.0)`) ; Pour créer le nœud `n2` avec la lettre `'c'`, sans `fils_droite`, un `fils_gauche` qui vaut `n1`, un nombre d'occurrences de 8 et une fréquence de 0.3, il suffit d'écrire `n2=Noeud(lettre='c','fils_gauche=n1,nb_occur=8,freq=0.3)` ou `n2=Noeud('c',n1,None,8,0.3)`.

Pour accéder aux champs d'un nœud, il suffit d'utiliser l'opérateur '.' : `print(n.lettre)` affiche le champs `lettre` du nœud `n` ; `print(n.fils_gauche)` affiche le champs `fils_gauche` du nœud `n` ;

`print(n.fils_droite)` affiche le champs `fils_droite` du nœud `n` ; `print(n.nb_occur)` affiche le champs `nb_occur` du nœud `n` ; `print(n.freq)` affiche le champs `freq` du nœud `n` ; Pour afficher un noeud `n`, il suffit d'écrire `print(n)`.

Pour tester l'égalité entre deux nœuds `n1` et `n2`, vous pouvez écrire `n1==n2` ou bien utiliser la fonction `comparenoeuds(n1,n2)`. Dans les deux cas, le résultat est `True` si les deux nœuds ont tous les champs égaux.

3 Questions

Question 1 : Dans le fichier `tp_huffman.py`, implémentez la fonction `arbre_exemple()` qui doit retourner le noeud correspondant à la racine de l'arbre donné ci-dessus dans le sujet. Par défaut, les fréquences seront initialisées à 0.0 mais les nombres d'occurrences seront respectés. Voici une trace d'exécution :

```
>>> arbre_exemple()
(.,(.,(E,None,None,8,0.0),(A,None,None,6,0.0),14,0.0),(.,(I,None,None,5,0.0),
    (.,(O,None,None,3,0.0),(U,None,None,2,0.0),5,0.0),10,0.0),24,0.0)
```

Question 2 : Dans le fichier `tp_huffman.py`, implémentez la fonction `construire_tableau_noeuds(texte)` qui lit dans le texte passé en paramètre tous les symboles présents dans le texte. La fonction retourne une paire constituée respectivement par :

- le nombre total de caractères dans le texte ;
- un tableau de nœuds où chaque nœud correspond à un symbole présent dans le texte. Le nombre d'occurrences du symbole est enregistré dans le champs `nb_occur` du nœud.
  Les nœuds n'ont pas de fils droite et gauche (les champs sont à `None`).
  La fréquence des nœuds (champs `freq`) est mis à 0.0.
  Les nœuds sont également ordonnés par ordre décroissant.

Autrement dit, la fonction retourne un tableau ordonné de nœuds correspondant aux nœuds du Pas 1 dans l'exemple si le texte est "AAAAAAEEEEEEEEIIIIIOOOUU" (compatible avec les nombres d'occurrences donnés dans l'exemple). Voici deux exemple d'exécution.

```
>>> longueur, tab_noeuds =  construire_tableau_noeuds("AAAAAAEEEEEEEEIIIIIOOOUU")
>>> print(longueur,":",tab_noeuds)
24 : [(E,None,None,8,0.0), (A,None,None,6,0.0), (I,None,None,5,0.0), (O,None,None,3,0.0),
 (U,None,None,2,0.0)]

>>> longueur, tab_noeuds =  construire_tableau_noeuds("Huffman")
>>> print(longueur,":",tab_noeuds)
7 : [(f,None,None,2,0.0), (H,None,None,1,0.0), (u,None,None,1,0.0), (m,None,None,1,0.0),
 (a,None,None,1,0.0), (n,None,None,1,0.0)]
```

Aide : utilisez par exemple un dictionnaire pour stocker dynamiquement le nombre d'occurrences des lettres rencontrées, puis utilisez la fonction `sorted` en vous inspirant de l'exemple ci-dessous.

```
>>> dico={'A': 22, 'B': 11, 'C': 33}
>>> sorted(dico.items(), key=lambda t: t[1],reverse=True)
[('C', 33), ('A', 22), ('B', 11)]
```

Question 3 : Dans le fichier `tp_huffman.py`, implémentez la fonction `frequence(tab_noeud,n)` prenant en paramètres un tableau de nœuds (`tab_noeud`) et un entier `n` et qui remplace toutes les

fréquences des nœuds présents dans le tableau par `nb_occur/n`, `nb_occur` étant le champs du même nom dans les nœuds du tableau. La fonction ne retourne aucune valeur. Voici une trace d'exécution :

```
>>> longueur, tab_noeuds =  construire_tableau_noeuds("Huffman")
>>> frequence(tab_noeuds,longueur)

>>> print("longueur=",longueur)
longueur= 7
>>> print("tab=",tab_noeuds)
tab= [(f,None,None,2,0.2857142857142857), (H,None,None,1,0.14285714285714285),
       (u,None,None,1,0.14285714285714285), (m,None,None,1,0.14285714285714285),
       (a,None,None,1,0.14285714285714285), (n,None,None,1,0.14285714285714285)]
```

Question 4 : Dans le fichier `tp_huffman.py`, implémentez la fonction `calcule_entropie(tab_noeud)` qui calcule l'entropie binaire associée aux fréquences des nœuds présents dans le tableau de nœuds `tab_noeud` passé en paramètre. Le logarithme en base 2 est donné par `math.log(x,2)`. L'entropie des fichiers `test.txt` (texte="Huffman"), `exemple.txt` (texte="AAAAAAEEEEEEEEIIIIIOOOUU") et `Huffman.txt` (long texte qui peut être chargé via la fonction `fichier_to_string(nom_du_fichier)`) sont respectivement 2.17, 2.52 et 4,37.

Question 5 : Dans le fichier `tp_huffman.py`, implémentez la fonction `construit_arbre_huffman(tab_noeud)` qui construit l'arbre de Huffman à partir des nœuds présents dans le tableau `tab_noeud` qui recensent toutes les lettres ainsi que leurs fréquences, triées par ordre décroissant des fréquences. La fonction retourne le nœud correspondant à la racine de l'arbre.
Contrainte : en cas d'égalité des fréquences, le nouveau nœud créé à chaque étape doit être placé après les nœuds existant et ayant la même fréquence.
Aide : à chaque étape, un nœud est créé et il doit être inséré dans le tableau de nœuds à la bonne position. Pour cela, insérez le nœud à la fin du tableau (`append`), puis utilisez la fonction `sorted` en vous inspirant de l'exemple ci-dessus.

Question 6 : Dans le fichier `tp_huffman.py`, implémentez la fonction `construit_dico_huffman(arbre,prefixe,dico)` qui construit un dictionnaire dont les clés sont les lettres situées aux feuilles de l'arbre passé en paramètre. La valeur associée à une lettre (une clé) est le chemin de la racine à la feuille (0 vers la gauche, 1 vers la droite) préfixé par le préfixe passé en paramètre.

Lorsque le paramètre préfixe est la chaîne vide `prefixe=''`, le code obtenu est le code de Huffman défini par l'arbre. Cette fonction peut être implémentée très simplement de façon récursive.
Aide : le nœud passé en paramètre est une feuille dès que son `fils_gauche` vaut `None`. Voici une trace d'exécution avec l'arbre de Huffman construit sur le texte `"Huffman"` :

```
>> dico={}
>> construit_dico_huffman(arbre,"",dico)
>> print("dico=",dico)
dico= {'f': '00', 'a': '010', 'n': '011', 'u': '100', 'm': '101', 'H': '11'}
```

Question 7 : Dans le fichier `tp_huffman.py`, implémentez la fonction `longueur_moyenne(arbre,h=0)` qui calcule la longueur moyenne d'un code de Huffman passé en paramètre via l'arbre de Huffman. Cette fonction prend en entrée un arbre représentant la racine d'un sous-arbre de l'arbre de Huffman et un paramètre `h` (la hauteur) indiquant la profondeur du sous-arbre dans l'arbre de Huffman. Le second paramètre est très utile pour une implémentation récursive de la procédure.

Les longueurs moyennes pour les fichiers `test.txt` (texte="Huffman"), `exemple.txt` (texte="AAAAAAEEEEEEEEIIIIIOOOUU") et `Huffman.txt` (long texte qui peut être chargé via la fonction `fichier_to_string(nom_du_fichier)`) sont respectivement 2.21, 2.57 et 4,40.

Question 8 : Dans le fichier `tp_huffman.py`, implémentez la fonction `codage(chaine, dico_codage)` qui encode la chaîne passée en paramètre à l'aide du dictionnaire `dico_codage`. La fonction retourne la chaîne encodée et le taux de compression donné par :

$$\text{taux} = \frac{\text{nb de bits de la chaîne initiale}}{\text{nb de bits de la chaîne codée}}.$$

On considérera que chaque caractère est codé sur 8 bits dans la chaîne initiale.