Codage de Huffman

T.P. VI

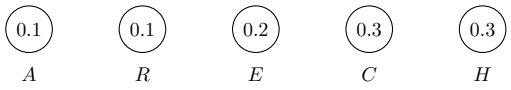
■ ■ ■

Partie I : Présentation de l'algorithme

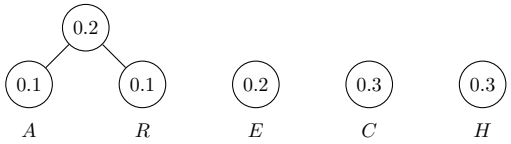
L'ensemble du T.P. est consacré à l'étude de l'algorithme de Huffman (1952) qui permet de coder un texte caractère par caractère à l'aide d'une chaîne binaire (i.e. une suite de 0 et de 1) en minimisant la longueur totale de la chaîne obtenue ; cet algorithme permet de faire de la compression de données. Par exemple, si la lettre e est plus fréquente que les autres lettres du texte, elle sera codée à l'aide d'un code plus court.

Exemple. Considérons le texte : "CHERCHECHA" que nous souhaitons coder avec des 0 et des 1. Le codage ASCII, par exemple, utilise 7 bits par lettre, soit un total de 70 bits pour ce mot. Le codage de Huffman que nous allons étudier ici permet de diminuer ce nombre de bits.
Ici, le texte est constitué des 5 lettres A, C, E, H, R, de fréquences respectives $(0.1, 0.3, 0.2, 0.3, 0.1)$.
On construit itérativement un arbre qui sera ensuite utilisé pour déterminer le code associé à chacune de ces lettres.

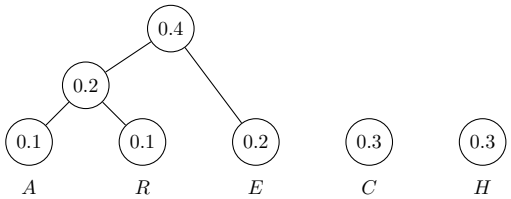
(i) On commence en représentant la fréquence de chacune des lettres.



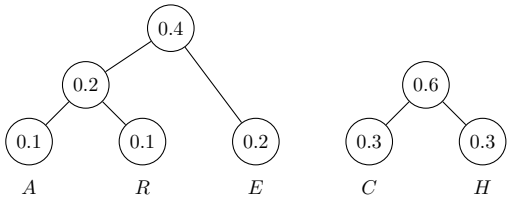
(ii) On regroupe deux des fréquences les plus faibles en un arbre au sommet duquel on inscrit la somme des fréquences. Les fréquences utilisées sortent alors du jeu, mais le nouveau somment entre dans le jeu, porteur de la somme des fréquences.



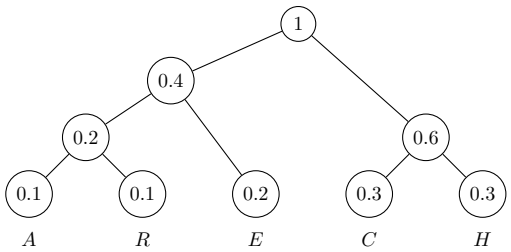
(iii) On continue en regroupant deux des fréquences les plus faibles.



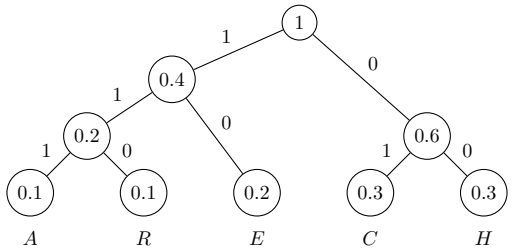
(iv) On continue en regroupant deux des fréquences les plus faibles.



(v) L'algorithme termine lorsque tous les nœuds initiaux sont reliés à l'arbre.



On affecte ensuite aux branches gauches de l'arbre des 1 et aux branches droites des 0. Le code d'une lettre est alors la succession des chiffres lus lors du parcours de l'arbre de la racine vers la lettre.



Ainsi, dans le cas présent,

$$A \to 111,\ R \to 110,\ E \to 10,\ C \to 01,\ H \to 00.$$

Le texte CHERCHECHA s'écrit alors 01001011001001001001111 qui contient 22 bits soit un taux de compression de $22/70 \approx 31\%$.
Vous pourrez remarquer que, contrairement au code MORSE, le code de HUFFMAN est un code préfixe, i.e. aucune lettre du code n'est préfixe d'une autre lettre.

Dans ce T.P., nous utiliserons la structure de **dictionnaire**. Un dictionnaire exprime une relation entre un ensemble de clés et un ensemble de valeurs.

- `{}` est le dictionnaire vide.
- `{k1:v1,...,kn:vn}` permet de construire explicitement un dictionnaire. Par exemple, `{'a' : 0, 'b' : 10}` est le dictionnaire contenant les clés `'a'` et `'b'`. Les valeurs associées à ces clés sont respectivement 0 et 10.
- `<dico>[<cle>]` permet d'accéder à la valeur associée à la clé `<cle>`.
- `<dico>[<cle>] = <valeur>` permet de créer la clé `<cle>` dans le dictionnaire `<dico>` et d'y affecter la valeur `<valeur>`. Si la clé était déjà présente, sa valeur est écrasée.
- `<cle> in <dico>` teste l'appartenance de `<cle>` au dictionnaire `<dico>`.
- `for <cle> in <dico>` permet d'itérer sur les clés.

1. Écrire une fonction `occurrences` qui, étant donnée une chaîne de caractères, renvoie les nombres d'occurrences des lettres de ce texte. Le résultat renvoyé sera de type dictionnaire où les clés sont les lettres présentes dans le texte et leur nombre d'occurrences.
Par exemple, `occurrences("abracadabra")` renverra `{'r':2, 'b':2, 'd':1, 'c':1, 'a':5}`.

Partie II : Construction du code de Huffman

Les arbres seront modélisés par des triplets dont le premier élément est la valeur du nœud, le second le sous-arbre gauche et le troisième le sous-arbre droit. Par exemple, l'arbre final représenté ci-dessus sera ainsi représenté par

$$(1, (0.4, (0.2, (0.1, A), (0.1, R)), (0.2, E)), (0.6, (0.3, C), (0.3, H))).$$

2. Écrire une fonction `extrait` qui, étant donnée une liste de tuples contenant au moins un élément, renvoie celui dont le premier élément est minimal et le supprime de la liste.
Par exemple, si `liste` contient `[(2, "a"), (3, (3, "b"), (0, "c")), (1, "e")]`, alors l'évaluation `extrait(liste)` renverra `(1, "e")` et modifiera `liste` en `[(2, "a"), (3, (3, "b"), (0, "c"))]`.

3. Écrire une fonction `unpas` qui, étant donnée une liste d'arbres contenant au moins deux éléments, renvoie ceux dont les deux premiers éléments sont minimaux et les supprime de la liste.
Par exemple, si `liste` contient `[(2, "a"), (3, (3, "b"), (0, "c")), (1, "e")]`, alors l'évaluation `unpas(liste)` renverra `((1, "e"), (2, "a"))` et modifiera `liste` en `[(3, (3, "b"), (0, "c"))]`.

4. Écrire une fonction `arbre` qui, étant donnée une liste de couples fréquence / lettre renvoie l'arbre décrit dans l'algorithme de Huffman.

5. Écrire une fonction `arbreverscode_aux(arbre, prefixe, dico)` qui, étant donnés un arbre, un prefixe déjà construit et un dictionnaire,

- si `arbre` est un couple fréquence/lettre, ajoute la clé lettre et la valeur prefixe à `dico` ;
- si `arbre` est un triplet `(n, a1, a2)`, appelle récursivement `arbreverscode_aux` sur `a1` et `a2` en modifiant les préfixes suivant la règle du code de Huffman.

La fonction suivante permettra alors, connaissant l'arbre de Huffman, de renvoyer le code correspondant.

```
def arbreverscode(arbre):
    return arbreverscode_aux(arbre, "", {})
```

6. Écrire une fonction `huffman` qui, étant donnée une chaîne de caractères, renvoie un dictionnaire permettant de coder chacune des lettres présentes dans cette chaîne en utilisant l'algorithme de Huffman.

Partie III : Taux de compression

7. Écrire une fonction `encode(texte, code)` qui, étant donnés une chaîne de caractères `texte` et un dictionnaire de code `code`, renvoie le texte codé.

8. Écrire une fonction `decode(texte_bin, code)` qui, étant donnés une chaîne de caractères en binaire `texte_bin` et un dictionnaire de code `code`, renvoie le texte décodé.

9. Écrire une fonction `code_ascii` qui, étant donnée une chaîne de caractères, renvoie le texte codé avec le codage ASCII.

On pourra utiliser les fonctions `ord` *et* `bin`.

Pour stocker la chaîne de caractère contenue dans le fichier `texte.txt` dans la variable `texte`, on peut utiliser le code suivant :

```
with open(texte.txt,"r") as f:
    Texte = f.read()
```

Pour supprimer les caractères qui ne sont pas représentés en `ASCII`, on peut utiliser la méthode `replace`.

10. Évaluer les taux de compression du code de Huffman par rapport au code ASCII sur les fichiers `06_camus.txt` et `06_rostand.txt`.

On remarque que le code dépend du texte codé. On devra ainsi transmettre le code en même temps que le texte codé. Cette technique est notamment utilisée dans le codage des images en `jpeg`.