# Rapport de travaux dirigés, Arbres de Huffman

TAMISIER Joshua - NADAUD Martin

January 9, 2026

---

## 1 Introduction

Dans ce TP, nous avons implémenté l'algorithme de Huffman dans le but d'encoder et de compresser des images ou du texte. Pour ce faire, nous avons en premier lieu implémenté l'algorithme de compréssion en python, puis nous l'avons adapté aux diverses utilisations possible.

# Contents

# 2 Algorithme de Huffman et codage source

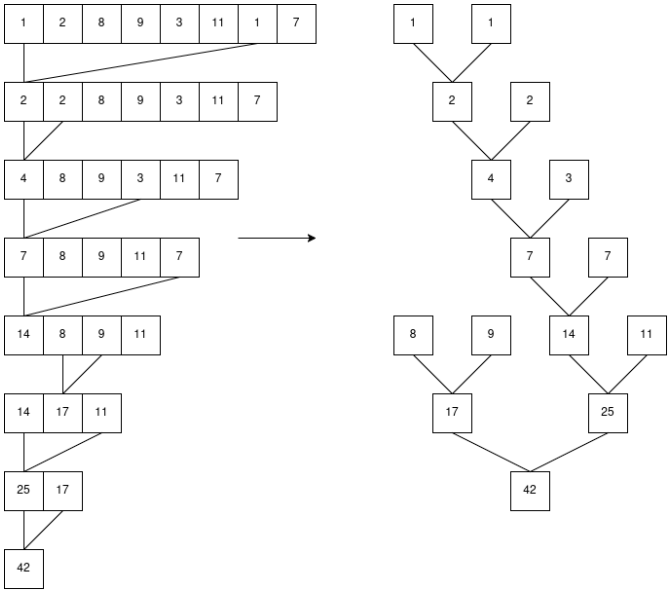
L'algorithme de Huffman est un algorithme de **codage source**. L'objectif est de réduire la longueur moyenne des mots de code. Voici le fonctionnement de l'algorithme de Huffman étape par étape.

## 2.1 Arbre de Huffman

La première étape est de construire un arbre binaire tel que la profondeur d'un mot[1] dans l'arbre soit inversement correllée a sa fréquence d'apparition.
Nous définissons un arbre de manière récursive en créant une classe `Noeud` avec deux enfants (droite et gauche) ainsi qu'une étiquette et un indice. L'algorithme utilisé fonctionne comme suit :

```
Entrée : p une Liste de nombre d'occurence de chaque mot de code.
Sortie : Arbre binaire de Huffman
Algorithme :
arbres = liste_de_nombres_vers_liste_de_feuilles(p)
tant que arbres contient au moins 2 éléments
    trier(arbres)
    nouvel_arbre = Noeud(
      enfant_droit  : avant_dernier(arbres),
      enfant_gauche : dernier(arbres)
    )
    arbres = concatener( tous_sauf_n_derniers(arbres, 2), nouvel_arbre)

retourner premier_élément(arbres)
```




Fonctionnement de l'algorithme de Huffman sur une liste d'entiers

[1] c'est a dire sa distance a la racine

Cet algorithme nous donne bien un arbre binaire, dont les feuilles correspondent aux éléments de la liste de départ, et dont la profondeur des feuilles est inversement correllée a leur valeur.

### 2.2 Nouveau code

De cet arbre, on peut définir une table de conversion `mot` $\rightarrow$ `code` de la façon suivante:

- On crée une liste des mots encodées, on concatene 1 pour l'enfant de droite, 0 pour l'enfant de gauche.
- On associe chaque code a un mot source en fonction de sa fréquence. Mot très fréquent $\rightarrow$ Code court[2].

[2]On remarque qu'il n'existe pas une seule et unique façon d'associer les mots et les codes. En effet, deux codes de même longueur peuvent être interchangé sans impacter la validité de l'algorithme.

# 3 Application

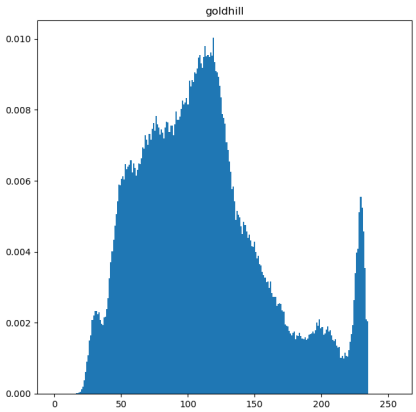
## 3.1 Compression d'images

### 3.1.1 Principe

On se propose d'appliquer l'algorithme de Huffman en compressant des images en niveaux de gris, qui seront représentées, à l'aide de la bibliothèque Image de Python, par des tableaux d'entiers compris entre 0 et 255. On définit donc une fonction `encoder_decoder(image)` qui encode puis décode une image avec les fonctions définies précedemment, teste si l'image décodée est égale à l'originale, puis affiche à l'écran les valeurs de l'entropie empirique, de la longueur moyenne et du ratio de compression. Pour une image donnée, on commence par afficher l'histogramme des valeurs prises, que l'on utilisera ensuite comme tableau de probabilités. On génère aussi un tableau d'entiers de 0 à 255 comme tableau des symboles. Pour respecter la précondition des fonctions de `huffman`, on doit trier préalablement le tableau des probabilités par ordre décroissant, en triant dans l'ordre correspondant le tableau de symboles. Pour encoder l'image on joint le tableau des "mots" encodés en binaire en une seule chaîne de caractères, et pour la décoder, on utilise la fonction `decode` et la liste des mots de code correspondants aux symboles, générée grâce à la fonction `huffman_code`. On affiche enfin à l'écran les valeurs demandées.

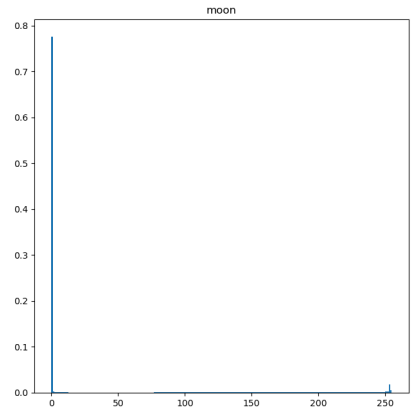
### 3.1.2 Expérimentations

(a) `goldhill.png`



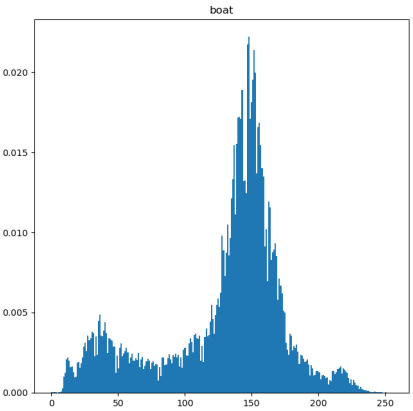

(b) Histogramme de `goldhill.png`

(a) `moon.png`




(b) Histogramme de `moon.png`

(a) `boat.png`




(b) Histogramme de `boat.png`

## 4 Pour aller plus loins