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.
Ici, nous gardon l'implémentation la plus générale possible. On ne sait pas quel type de données on encode, mais l'algorithme fonctionne de la même manière pour tout type.

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)
```

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

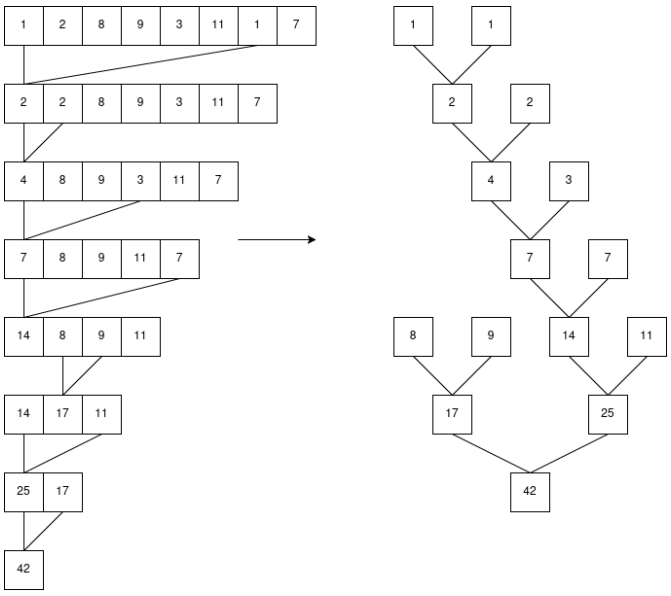
De cet arbre, on peut définir une table de conversion `mot` $\rightarrow$ `code` de la façon suivante:

- 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].

Grâce a cette table d'association, on peut alors encoder un à un les mots de la source, afin d'obtenir le message encodé.

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

[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.



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

2.3 Décodage

Une fois encoder, on peut décoder les données en utilisant le même arbre. Les codes de Huffman étant préfixes, le décode peut se faire de manière "gloutonne". Le décodage se fait en prenant chaque bit l'un après l'autre, en déçandant l'arbre jusqu'a arriver a une feuille. Par construction, on a la certitude que cette feuille correspond bien au mot encodé.
Ainsi, notre algorithme de décodage fonctionne ainsi :

```
Entrée : seq une séquence de bits représentant les mots codés,
         tree l'arbre utilisé pour les encoder.
Sortie : une séquence de mots décodés
Algorithme :
offset := 0
noeud_curseur := tree
décodé := []
tant que seq n'est pas vide:
  si est_feuille(noeud_curseur) :
    seq = enlever_n_premier_bits(seq, longueure)
    décodé = concatener(décodé, mot(noeud_curseur))
    longueure = 0
    noeud_curseur = tree
  sinon
    si seq[0] = 0 :
      noeud_curseur = noeud_curseur.enfant_gauche
    sinon :
      noeud_curseur = neoud_curseur.enfant_droit
    longueur = longueur + 1

retourner décodé
```

3 Application

4 Pour aller plus loins

La question est, peut on améliorer cet algorithme. On sait que les codes de Huffman sont optimaux, et donc le seul moyen d'améliorer le ration de compréssion serait de réduire l'entropie de la source. Voici donc une proposition d'algorithme pour réduire l'entropie de la source avant de la compresser par huffman.

4.1 Proposition d'algorithme

L'idée de cet algorithme consiste a remplacer la valeur de chaque pixel par la différence entre lui et le pixel précédant. C'est ce qu'on appel du `Delta Encoding`. Voici un exemple de code C pour encoder un tableau ainsi :

```
void encode(static char* input, int input_len, char* output)
{
  //On suppose que output est initialisé a 0
  output[0] = input[0];
  for(int i = 1; i < input_len; i++)
  {
    output[i] = input[i] - input[i-1];
  }
}
```

Si une image n'est pas aléatoire, c'est a dire qu'elle contient des patternes (comme par exemple une photographie), on peut s'attendre a des différences similaires a plusieurs endroits de la photo. Par exemple, avec cet encodage préalable, tous les aplats de couleurs seront représentés par des 0. On peut donc s'attendre a une entropie réduite, et donc a un code de Huffman plus éfficace. Cette méthode est nottament utilisé dans les images aux format `png` [3].

[3] https://www.w3.org/TR/png-3/#7Filtering