COMPRESSION DE DONNEES I

Code Optimal:

Def: Code de la plus petite longueur moyenne avec l'entropie moyenne minimale.

↳ **Lemme:** dans un tel code: mot de code le ⊕ long associé au message le moins probable, et les deux mots les plus longs se distinguent uniquement par le dernier caractère.

taux de compression d'un message: $\frac{\text{longueur du message codé}}{\text{idem, par le code de longueur fixe}}$

I) Code de Huffman:

Objectif: réduire la longueur du code en utilisant les freq° d'app°

Principe: Construire un arbre de Huffman de manière ascendante, et coder les symboles grâce à lui.

Variantes:

↳ méthode non-adaptative: table des fréq° vient de l'extérieur.
↳ méthode semi-adaptative: table des fréq° établie par lecture préalable.
↳ méthode adaptative: table des fréq° construite au f. et à m. de la compression à partir des données traitées.

Rq:
- code "Entropique" (basé sur statis d'apparition)
- code non destructif
- arbres finaux non uniques, mais tous de m longueur moyenne.
- taux de compression entre 30% et 60%.
- aucun codage déchiffrable dans ce type de compression de longueur moyenne inférieure.
- décodage via dictionnaire envoyé en ⊕: taille du message augmentée.

II Méthode semi-adaptative :

↳ non adaptative marche pareil

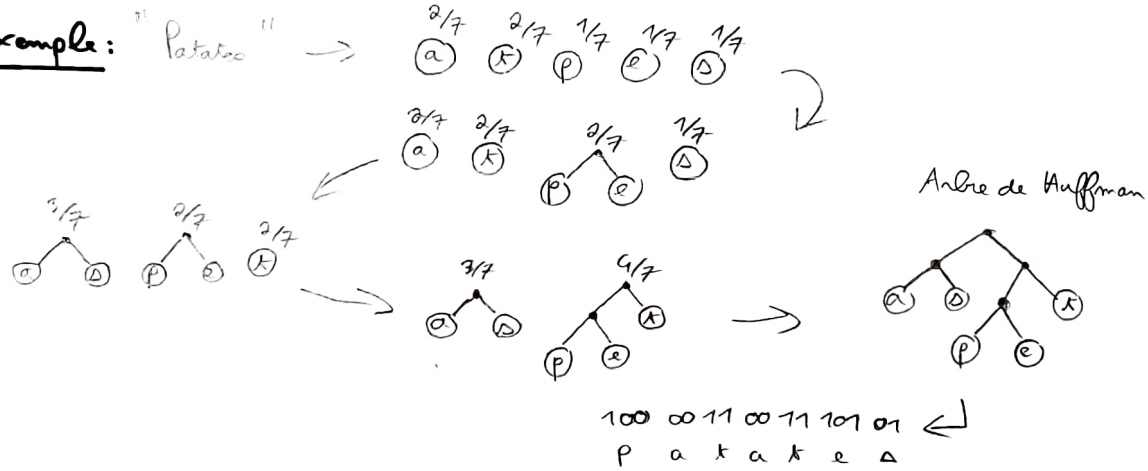
Etapes : ① Initialisation : on lit le message pour construire la table des fq.

② Etape initiale : pour chaque élément de la source, on crée un arbre réduit à sa racine avec pour valeur sa fréq° d'apparition.

↻ ③ Réduction : On fusionne les deux arbres de ⊕ petites valeurs. nouvelle valeur = somme des anciennes.

④ Attribution : On attribue des valeurs / mots de code aux feuilles selon le chemin pris pour les atteindre. (Ex : 0 gauche, 1 droite)

Exemple : "Patatee" →



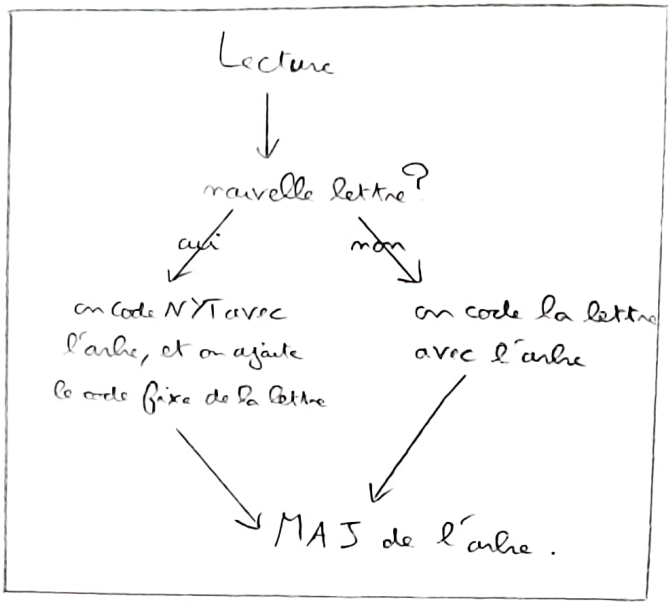
III Méthode adaptative :

a) NAI de l'arbre : - contient une feuille NYT, toujours fils gauche.
- tq un noeud a toujours Fils gauche ≤ fils droit.
- les noeuds sont les sommes des fils, les feuilles sont les lettres et leur nombre d'apparition.

Quand on trouve une lettre, la feuille NYT devient un arbre avec racine, NYT comme fils gauche et la nouvelle lettre comme fils droit. Ensuite, on équilibre l'arbre pour qu'il respecte les règles ↑.

B) Encodage : Algo :

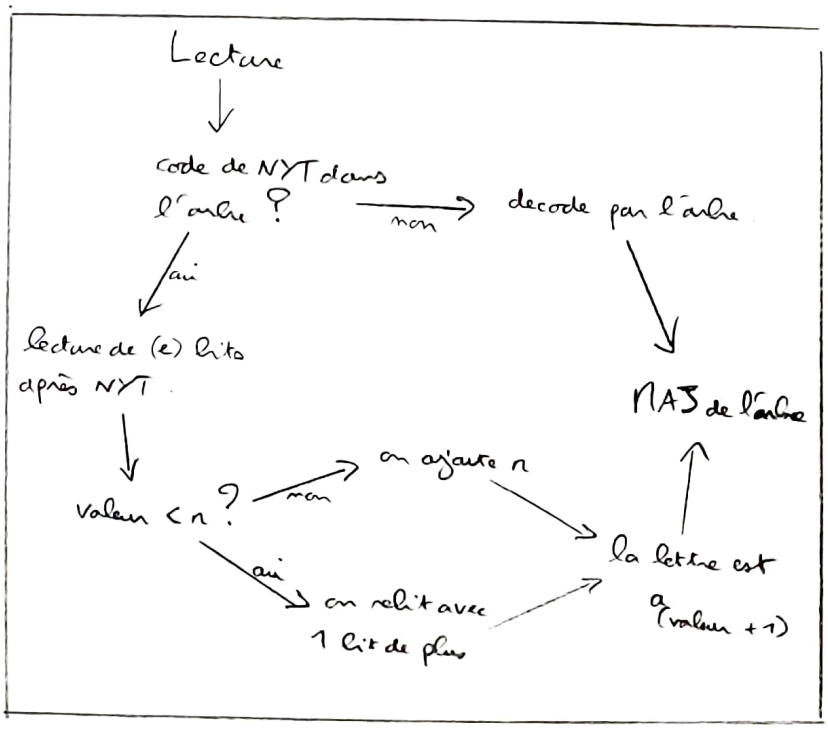
↳ paramètres :
- ↳ $m = |\text{alphabet}|$
- ↳ (e, r) tq $\begin{cases} m = 2^e + r \\ 0 \leq r \leq 2^e \end{cases}$
- ↳ a_k est la k-ième lettre de l'alphabet.



Code fixe : a_k est encodé par :

$\begin{cases} 1 \leq k \leq 2r \rightarrow (k-1) \text{ en binaire sur } (e+1) \text{ bits} \\ k > 2r \rightarrow (k-r-1) \text{ en binaire sur } e \text{ bits} \end{cases}$

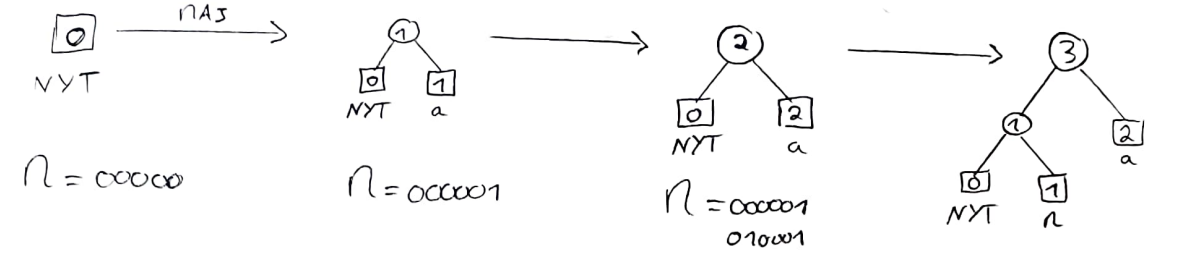
C) Décodage : Algo :



↳ **Rq :** On construit l'arbre pour le codage **et** le décodage.
Naïb pas besoin de transmettre un dico, juste (e) et (r).
(pour l'alphabet latin, $m = 26 = 2^4 + 10$)

Exemple : " aardvark " qu'on codera dans le message Ω.

$m = 26 = 2^4 + 10 \rightarrow \begin{cases} e = 4 \\ r = 10 \end{cases}$.

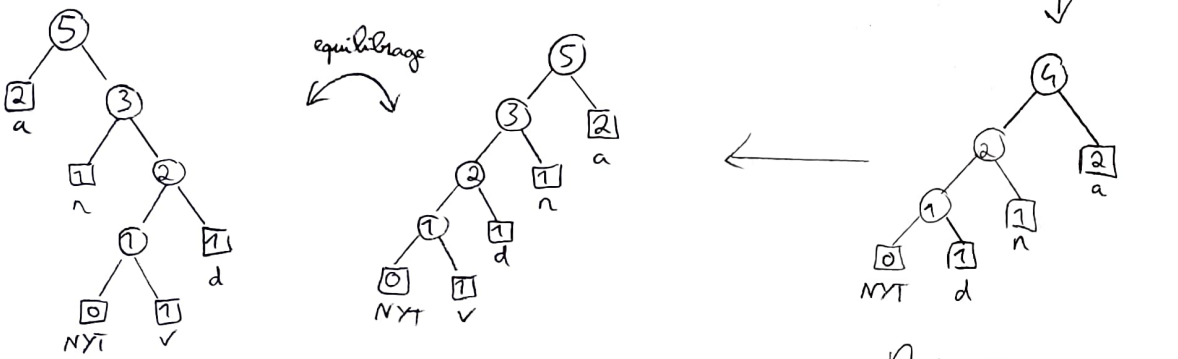


$\Omega = 00000$

$\Omega = 0000001$

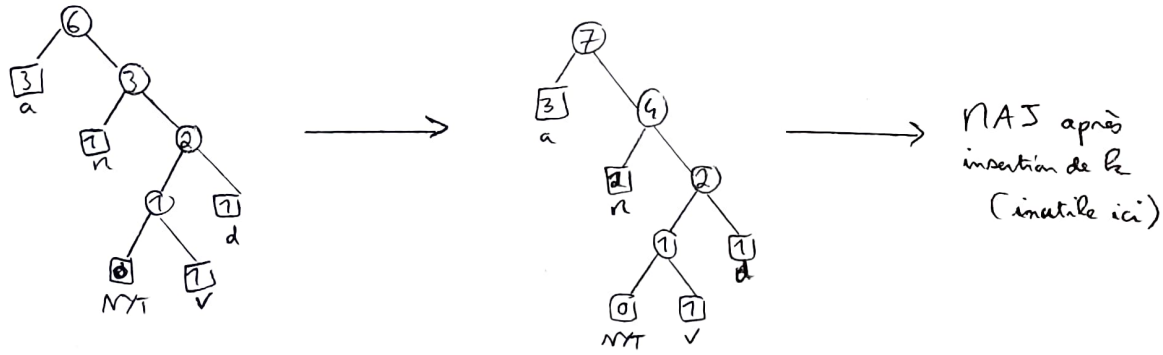
$\Omega = 0000001$
010001

$\Omega += 00000111$



$\Omega += 0001011$

$\Omega += 0$



NAJ après insertion de la (inutile ici)

$\Omega += 10$

$\Omega += 1100010100$

mot final : $\Omega =$ " 00000101000100000111000101101011000101010 "

↳ voir vidéos "CSE concepts with Parinita" pour plus d'explications.