Binôme :AB : Bouchhioua Wejdane &Timofte Béatrice

Récapitulatifs des Tps Informatique

Remplir la colonne « état » de chacun des items des Tps.
A : Mieux que demandés et/ou exercices facultatifs fait. B : Testé et validé.
C : Fait mais moins bien que demandé. D : Fait mais Buggé. E : pas fait !!

TP1	Etat A,B,C, D,E	Commentaires et questions
Chiffreur par décalage	B	Adaptation de la classe mère CodeurCA
IFrequences	B	Les fréquences sont calculées à partir de la fréquence courante
afficheTableauDesFrequences	A	
afficheTableauDes FrequencesDecroissantes	A	Utilisation de la fonction « sorted » présente dans la démonstration de cours
afficheHistogrammeDesFrequences	A	Implementation de la bibliothèque matplotlib.pyplot
Histogramme des livres de littérature et des images à mettre dans la démo	A	Remarque les histogrammes sont semblables pour chaque type de fichiers exemple les histogrammes des fichiers textes des livres sont semblables et les images sont semblables aussi . Voir Annexe pour les résultats des histogrammes.
Histogramme des DocChiffre1, DocChiffre2, DocChiffre3, DocChiffre4	B	Nous disposons que du DocChiffre1 dans le dossier des fichiers joints.
TP2		
Déchiffrement de :		
Chiffre1		??
Chiffre2		??
Chiffre3		??
Chiffre4		??
Chiffreur Affine	A	La classe ChiffreurAffine prend deux paramètres a et b lors de l'initialisation, représentant les coefficients pour la

		transformation affine.Elle inclut les méthodes de cours pour l'encodage binaire (binCode) et le décodage (binDecode) en utilisant la transformation affine spécifiée. Il y a la méthode de démonstration demo() qui montre comment l'encodage et le décodage fonctionnent pour différentes instances.
Fonction ChiffreurParVal	B	On a calculé les coefficients pour le déchiffrement. Dans la variable a_inv on crée un objet de la classe ElementDeZnZ avec b1 et le modulo 256, puis on appelle la méthode inverse() pour calculer l'inverse de b1 modulo 256. Cela donne l'inverse multiplicatif de b1 dans l'anneau Z/256Z. On crée un nouvel objet ChiffreurAffine avec les coefficients a et b calculés et on le retourne
Chiffreur Vigenere	??	il n'y a pas dans l'énoncé de TP
fBijParDecallage	A	Implémentation de la classe fBijParDecallage héritant de Fbijection8bitsCA, en utilisant l'implémentation de la méthode donnée dans la classe mère pour faire les graphiques.
Test sur des bijections	A	Bon fonctionnement pour les 3 cas de test : fonction affine, bijective inventée(returner la valeur + le décallage, en modulo 256), fonction en appliquant un masque.
Schéma de Feistel avec son test	A	Reprise de la classe FFeistel8bits qui hérite de la classe FBijection8bitsCA vue en cours. On a testé en affichant les graphiques de diffusion-confusion en appelant la methode afficheGraphiquesDeDiffusionConfusion() de la classe mère.
ChiffreurECB	A	ECBChiffreur est une classe qui implémente le mode de chiffrement Electronic Codebook (ECB). Avec La méthode chiffrer il prend un message en entrée et le divise en blocs de 8 octets. Elle chiffre ensuite chaque bloc indépendamment en appliquant la bijection à chaque octet du bloc. Les blocs chiffrés sont ensuite concaténés pour former le message chiffré final.

ChiffreurCBC	A	CBCChiffreur est une classe qui implémente le mode de chiffrement Cipher Block Chaining (CBC). La méthode chiffrer prend un message en entrée et le divise en blocs de 8 octets. Elle chiffre chaque bloc en effectuant un XOR (la bijection choisie) avec le bloc chiffré précédent (ou le vecteur d'initialisation pour le premier bloc), puis applique la bijection à chaque octet du résultat. Les blocs chiffrés sont concaténés pour former le message chiffré final.

TP3	Etat A,B,C, D,E	Commentaires et questions
liste16Cles	A	OK pour l'implémentation de la fonction qui génère les 16 clés de sous-rondes à partir de la clé principale spécifiée.
lbSImage	A	Fonctionnel et applique la S-Box spécifiée sur une liste de 6 bits.
functionF	D	Doit implémenter la fonction F du DES, qui inclut l'application des S-Boxes et la permutation P.
lbround	D	Doit appliquer un round au bloc de bits en utilisant la clé spécifiée et la permutation P.
FBij64BitsDES	C	Une classe abstraite de bijection
ChiffreurDES	C	Hérite de la classe CodeurCA et utilise la classe FBij64BitsDES pour effectuer le chiffrement et le déchiffrement.
Optionnel : Chiffreur DES CBC	E	

TP4	Etat A,B,C,	Commentaires et questions

	D,E	
EstPremierOuPseudoPremierAvecA(n,a)	A	Renvoie le test de la pseudo-primalité d'un entier n selon Miller-Rabin avec le nombre a => True si n est probablement premier ou pseudo-premier avec la base a, False sinon.
Premier faux positif avec 2	A	Elle utilise les concepts de décomposition en facteurs premiers et de résidus quadratiques modulo n pour déterminer la primalité.
EstPremierOuPseudoPremierAvecLA(n,la)	A	Cela étend le test précédent en vérifiant la primalité de n avec une liste de bases la. Elle teste si n est probablement premier ou pseudo-premier avec toutes les bases de la liste LA. Cela permet une meilleure robustesse dans la détermination de la primalité.
Premier faux positif avec [2,3]	B	Renvoie un résultat faux
Premier faux positif()	C	
Fonction estPremier	A	Nous implémentons un autre test de primalité probabiliste de Miller-Rabin qui teste si n est premier en utilisant k tests aléatoires. C'est une version plus robuste du test Miller-Rabin.
Calcul de clés pour RSA	A	generer_cles_RSA(taille), nous avons implémenté cette fonction pour générer une paire de clés RSA, une clé publique et une clé privée. La taille spécifiée est la taille en bits du module RSA. Elle utilise des sous-fonctions pour générer des nombres premiers, choisir un exposant public et calculer l'inverse modulaire.
ChiffreurRSA	A	Cette fonction chiffre un message donné en utilisant la clé publique RSA fournie. Elle convertit chaque caractère du message en sa représentation numérique, puis élève ces nombres à la puissance de la clé publique modulo n.

TP5	Etat A,B,C, D,E	Commentaires et questions
ElemtE07 init et add	A	Les fonctions sont complétées en respectant les formules de l'addition et l'initialisation vérifie les insertions d'éléments
ElemtE07 double et mul	A	Les fonctions sont adaptées avec les

		formules données en cours double 2.A et pmu pour la multiplication en cas général k.A
ElemtE07 Autres méthodes	A	Toutes les fonctions sont implémentées et nous avons le résultat de la courbe elliptique. La courbe est jointe en annexe
ChiffreurE07	D	Nous rencontrons un problème avec le constructeur init de ChiffreurE07 qui cause des erreurs
Recherche de collision	A	Nous avons simulé la recherche de collision dans une liste donnée de fichiers MP3, en prenant en entrée une liste de noms de fichiers MP3. La fonction recherche_collision(liste) choisit aléatoirement un fichier de la liste et vérifie s'il a déjà été sélectionné précédemment. Si c'est le cas, la fonction renvoie le nombre total de tirages nécessaires pour obtenir une collision. La fonction statistique (taille_liste, nb_simulations), effectue des simulations pour chaque taille de liste de fichiers MP3 et calcule la moyenne du nombre de tirages nécessaires pour obtenir une collision. La méthode generer_graphique(taille_max_liste, pas, nb_simulations) génère un graphique illustrant comment le nombre moyen de tirages pour obtenir une collision varie en fonction de la taille de la liste de fichiers MP3.

TP 6	Etat A,B,C,D,E	Commentaires et questions
Recherche de collision - paradoxe anniversaires	A	Nous avons simulé le Paradoxe des Anniversaires et généré un graphique illustrant la probabilité de collision d'anniversaires en fonction du nombre de personnes présentes. Nous avons tracé un graphique illustrant la probabilité de collision des anniversaires en fonction du nombre de personnes, en utilisant une plage de nombres de personnes spécifiée et un nombre de simulations.

		Les paramètres de la simulation sont définis avant d'appeler la fonction plot_birthday_paradox pour exécuter la simulation et afficher le graphique des résultats.
Fonction de Hachage par Davis-Meyer	A	ChiffreurNaif : Cette classe représente un chiffreur naïf qui chiffre et déchiffre les données en utilisant l'opération XOR. Les méthodes binCode et binDecode effectuent respectivement le chiffrement et le déchiffrement en appliquant l'opération XOR sur les bits des données. La fonction de hachage Davies-Meyer utilise le chiffreur naïf pour transformer le message en un hash en suivant le schéma Davies-Meyer.
Test d'une fonction de hachage naïve	A	Notre script teste ensuite la fonction de hachage Davies-Meyer avec quelques exemples de messages, affichant les hash résultants. Enfin, on affiche un histogramme des fréquences de hashage pour les exemples.
Test de SHA256 sur la fréquence du premier octet avec le Xhi²	A	L a fonction chisquare de la bibliothèque scipy.stats est utilisée pour effectuer le test du Xhi². Les fréquences observées des premiers octets (first_byte_frequencies) sont comparées aux fréquences attendues pour une distribution uniforme (expected_frequencies). t.
Test de SHA256 sur la fréquence des ordres des trois premiers octets avec le Xhi²	A	Les fréquences observées des ordres des trois premiers octets (byte_order_frequencies) sont comparées aux fréquences attendues pour une distribution uniforme (expected_frequencies_order).

TP 7	Etat A,B,C,D,E	Commentaires et questions
Création et test (méthodes du TP6) d'une fonction de hachage naïve	A	Nous avons créé une fonction de hachage naïve qui calcule le hash d'une chaîne de données en utilisant l'opération XOR. La méthode binCode effectue le calcul du hash en prenant chaque byte de la chaîne et en effectuant un XOR successif sur eux pour obtenir la valeur de hash. Le script teste ensuite la fonction de hachage naïve en générant une chaîne de test en ajoutant différentes combinaisons de caractères à la fin d'une chaîne principale. Il calcule le hash de chaque chaîne de test et stocke les valeurs de hash dans une liste. Enfin, il affiche un histogramme des fréquences de hashage pour les chaînes de test générées.
Fonction de Hachage par Davis-Meyer	A	Nous avons développé les fonctions hachage par Davies Meyer en utilisant une fois le hachagenaïff avec XOR et en utilisant le chiffrement par DES sur 64 bits
Tests de cette fonction de hachage	A	Nous avons réussi à effectuer les tests pour les deux méthodes de hachage en générant les histogrammes de fréquences pour la hachage utilisant le chiffrement natif et le hachage utilisant un chiffrement plus solide qui est le DES . Les deux histogrammes ont une distribution totalement différente, le hachage par DES dispose d'une distribution plus importante. Les histogrammes sont joints dans l'annexe.

TP 8	Etat A,B,C,D,E	Commentaires et questions
Calcul d'Entropie	A	Nous avons implémenté les fonctions "entropie" et "entropieDictionnaire" permettent de calculer l'entropie d'une séquence de données binaires et d'un dictionnaire de deux octets

		respectivement. La fonction entropie parcourt la séquence d'octets, calcule la fréquence d'apparition de chaque octet, puis utilise la formule de l'entropie pour déterminer la quantité d'information moyenne contenue dans la séquence. Quant à la fonction entropieDictionnaire, elle analyse les paires d'octets consécutives dans la séquence, compte leur fréquence d'apparition, puis calcule l'entropie basée sur ces occurrences.
Compression par répétition	A	classe RepetitionCompressor qui hérite de CodeurCA .La compression par répétition est une méthode où les séquences de bits identiques consécutives sont remplacées par une paire (count, bit).
Test des compressions	A	Pour effectuer les tests nous avons créé une instance du compresseur par répétition (RepetitionCompressor), qui itère à travers cinq motifs binaires différents (générés par LBinaire603.exBin603) puis affiche le binaire d'origine, le binaire compressé, et vérifie si la décompression du binaire compressé est égale au binaire d'origine.
CompresseurRLE (facultatif)	A	RLE est une sous-classe de CodeurCA et implémente l'algorithme de compression/décompression RLE. Pour compresser elle prend un objet LBinaire603 (monBinD) en entrée, initialise un objet LBinaire603 vide (compressed_data) pour les données compressées puis itère à travers les données d'entrée en commençant par le deuxième bit.Si le bit courant est égal au précédent, elle incrémente count. Si le bit courant est différent, ajoute la séquence (count, bit courant) aux données compressées, réinitialise count et met à jour current_run.
CodageHuffman	A	Le Codage Huffman consiste, d'après la fonction arbreDepuisListePonderee, à créer un arbre binaire où les symboles les plus fréquents sont placés près de la racine, tandis que les moins fréquents sont situés aux extrémités. Les symboles

		sont ensuite encodés en utilisant des chemins spécifiques dans cet arbre.
Compresseur Huffman	A	Le compresseur Huffman (binCode) prend en entrée un dictionnaire de données et le compresse en utilisant les codes binaires générés par le Codage Huffman. Cela permet de réduire la taille des données en remplaçant les symboles fréquents par des codes courts et les symboles moins fréquents par des codes plus longs.
Le premier Codeur correcteur d'erreurs	A	Nous avons implémenté la fonction TestErreur qui vérifie les sommes des lignes et des colonnes dans une matrice carrée de 64+8+8 octets. Elle prend une liste appelée matrice en entrée. Avec l'utilisation de deux boucles de vérification des sommes des lignes et des sommes des colonnes elle calcule la somme des octets puis compare la somme. Si une erreur est détectée, la fonction renvoie un message indiquant la ligne ou la colonne où l'erreur a été trouvée. La fonction CodeurCA prend une liste de listes appelée LBinaire603 en entrée, où chaque liste représente une matrice de 64 octets, calcule les sommes des lignes et des colonnes pour chaque matrice, puis ajoute les sommes des lignes et des colonnes à la fin de chaque matrice avec la méthode extend

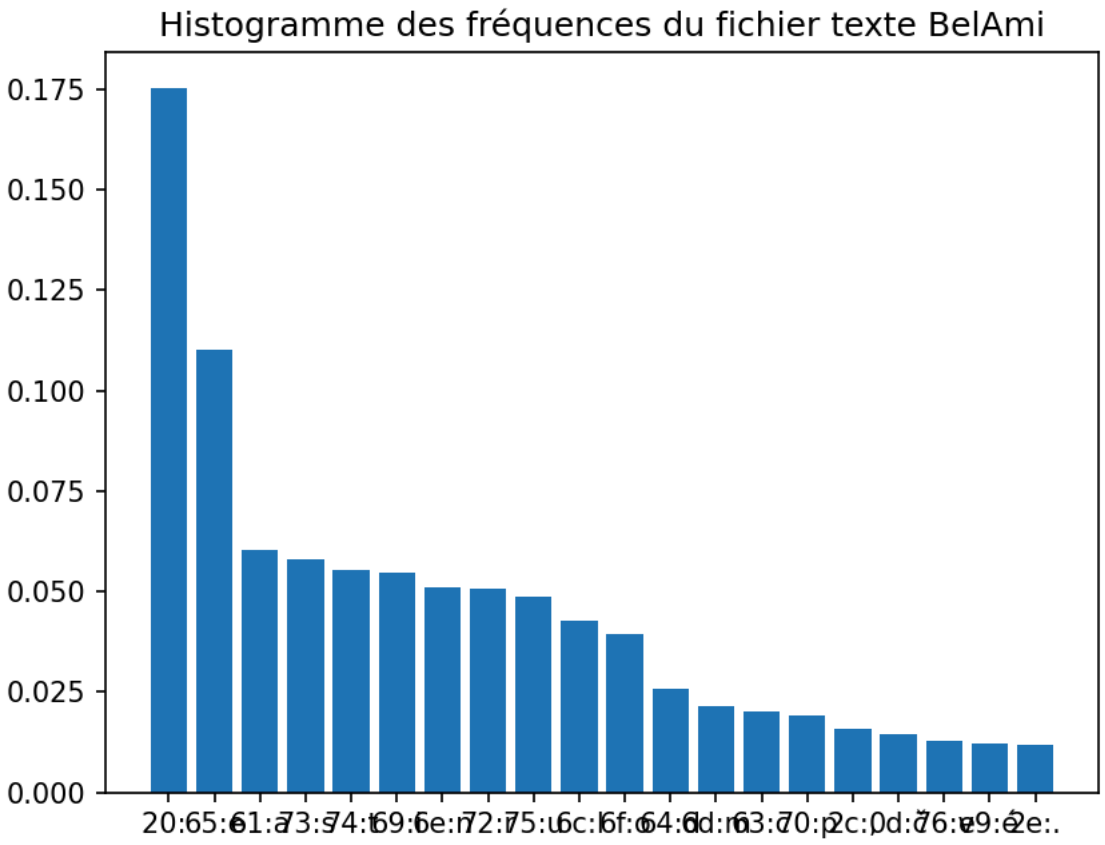
TP9 et 10	Etat A,B,C, D,E	Commentaires et questions
CodeurParParité avec correction d'erreur	A	• L'objectif de ce codeur est de prendre des données d'entrée de 49 bits et de les encoder en une séquence de 64 bits, en ajoutant des bits de parité. • Pour cela, le codeur calcule la parité des 49 bits d'entrée, c'est-à-dire le nombre de bits à 1. Si le nombre de bits à 1 est pair, le bit de parité est 0, sinon il est 1. • Ensuite, le codeur ajoute ce bit de parité à la fin des données d'entrée pour obtenir une séquence de 50 bits. • Enfin, il complète la séquence à une taille de 64 bits en ajoutant les mêmes bits de parité jusqu'à atteindre cette taille.
Class PolF2	A	Donnée par le professeur, nous avons développé la fonction XOR() dans la classe PolF2, qui permet de faire l'opération OU EXCLUSIF entre deux objets de type PolF2. Cette fonction sera nécessaire par la suite pour la méthode BlocCode() de la classe de codeur CRC8.
Codeur CRC8 avec correction d'erreur	A	Le codeur CRC8 est une classe Python qui prend en charge le codage de blocs de 32 bits avec une correction d'erreur CRC sur 8 bits. Il utilise un polynôme générateur (Pg) pour effectuer le codage. La méthode blocCode prend un mot de 32 bits en entrée et retourne un mot de 40 bits avec le CRC8 ajouté à la fin. Si la longueur du

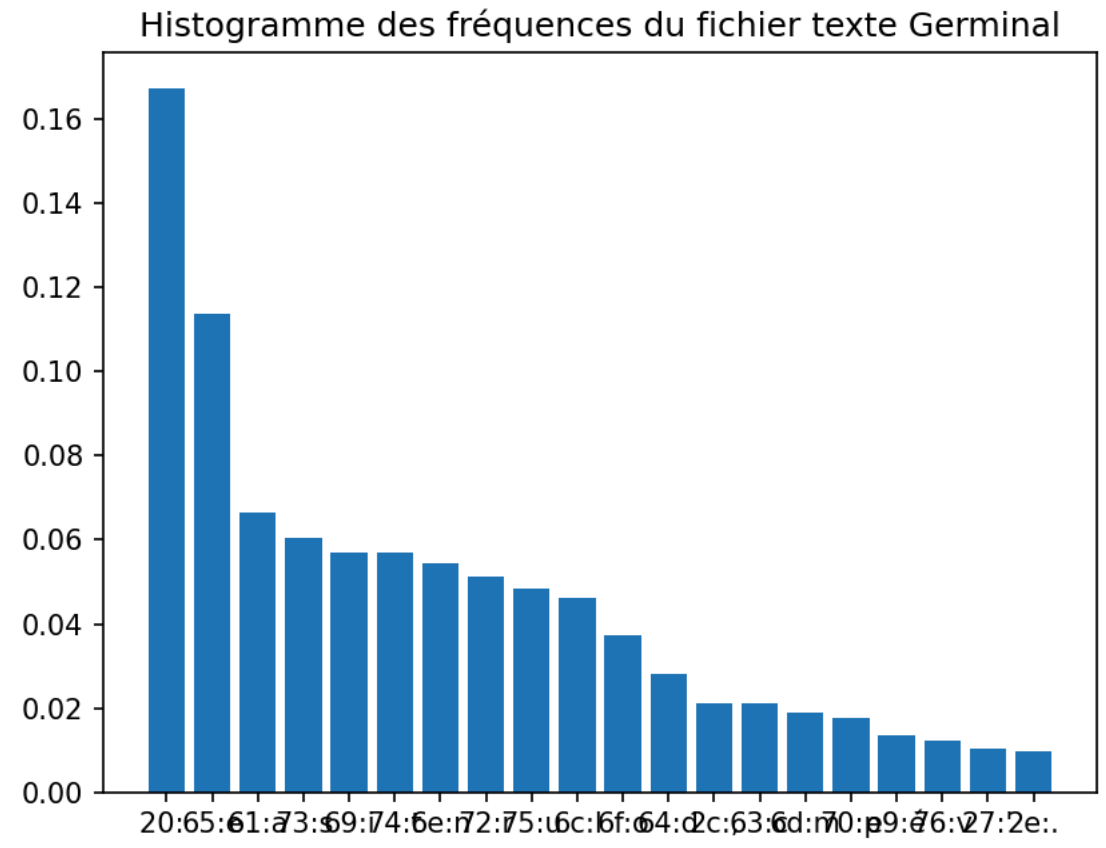
		message n'est pas un multiple de 4, elle est complétée pour avoir une liste de 4*N. La méthode estBlocValide vérifie si un bloc de 40 bits est valide en effectuant une division par le polynôme générateur. La méthode blocValideLePlusProche trouve le bloc valide le plus proche pour un bloc donné avec des erreurs.
Décodeur CRC8 avec correction d’erreur	B	La méthode blocDecode décode un bloc de 40 bits et vérifie s'il contient des erreurs en effectuant une division par le polynôme générateur. Si le bloc est valide, il est renvoyé sous forme de mot de 32 bits. Sinon, une erreur est levée. La méthode binDecode décode un message binaire complet composé de plusieurs blocs de 40 bits.

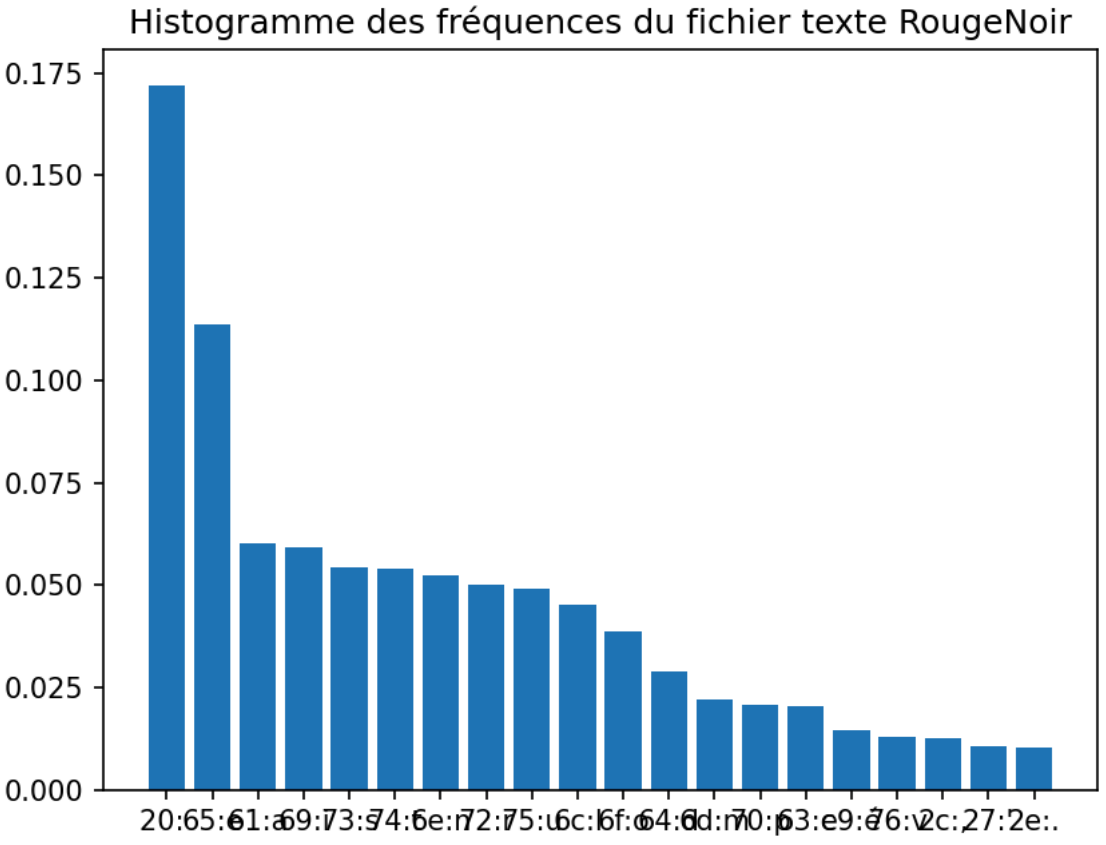
ANNEXE

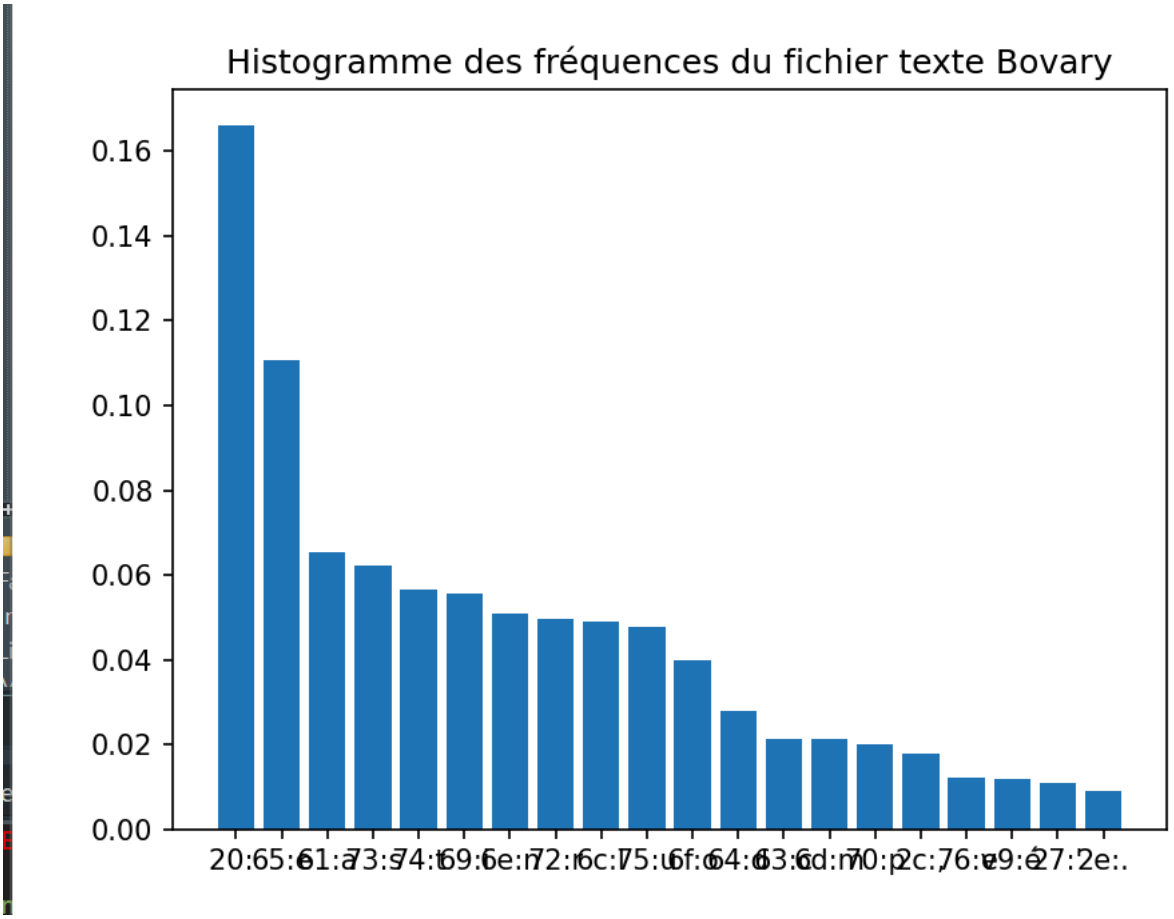
TP1 :

HISTOGRAMMES FICHIERS LIVRES :

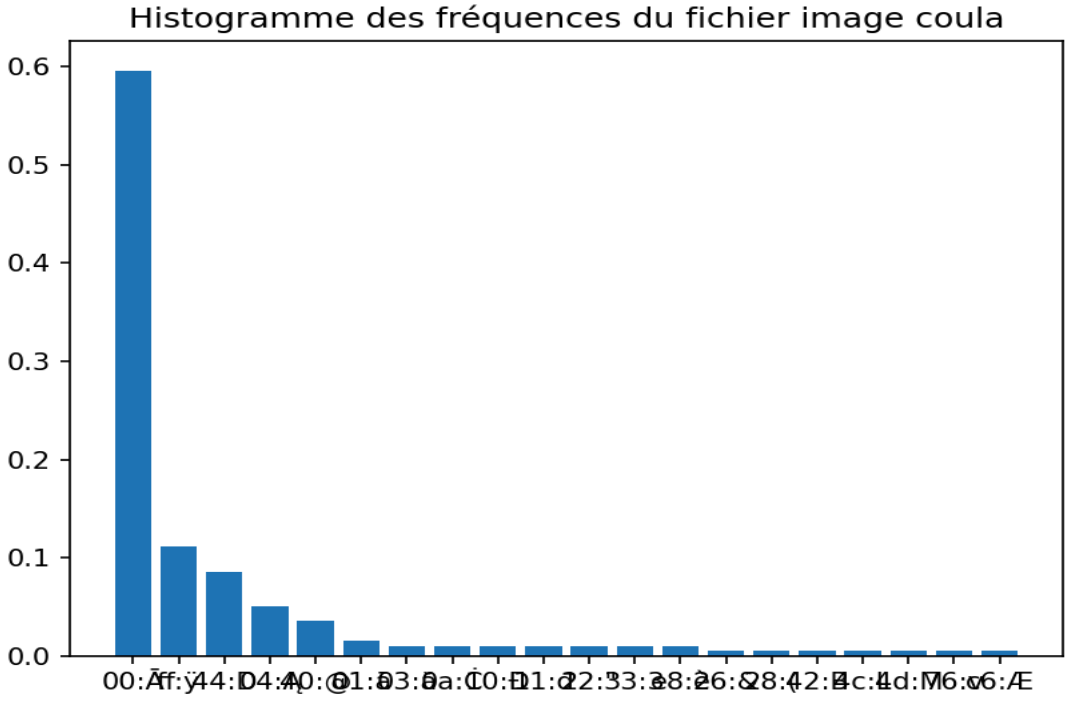


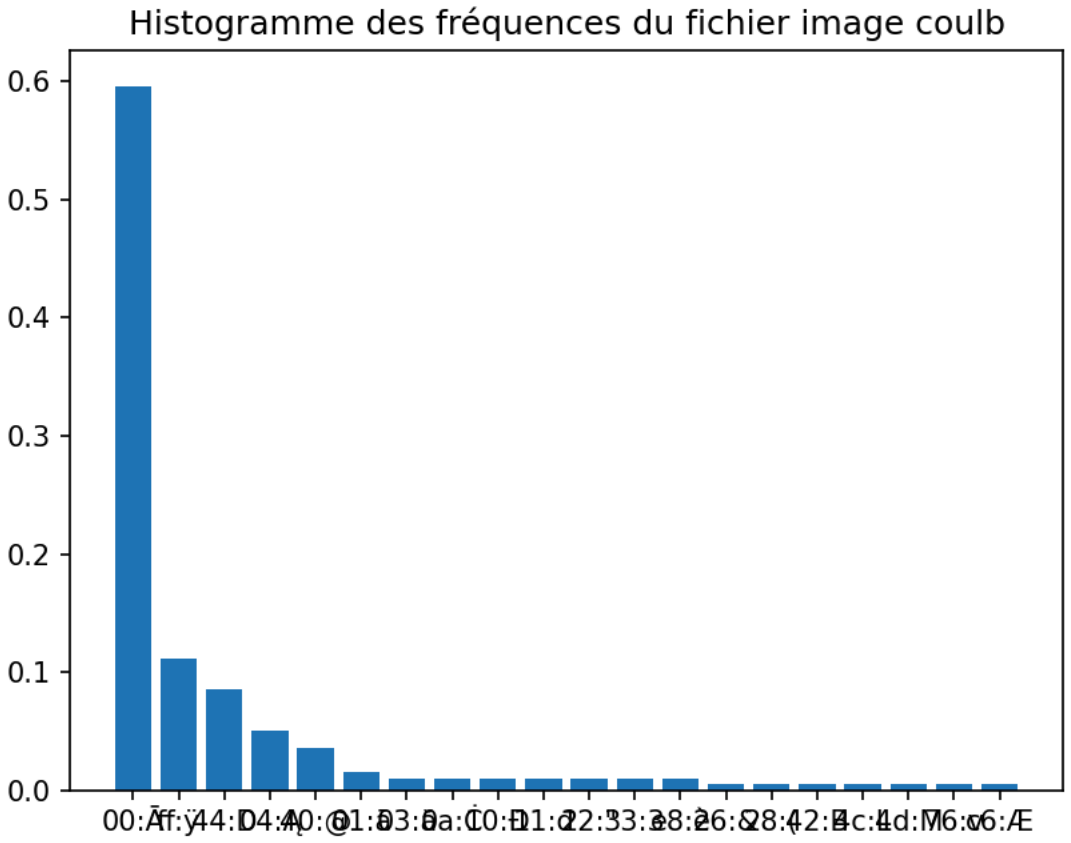

Histogramme des fréquences du fichier texte Germinal
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00

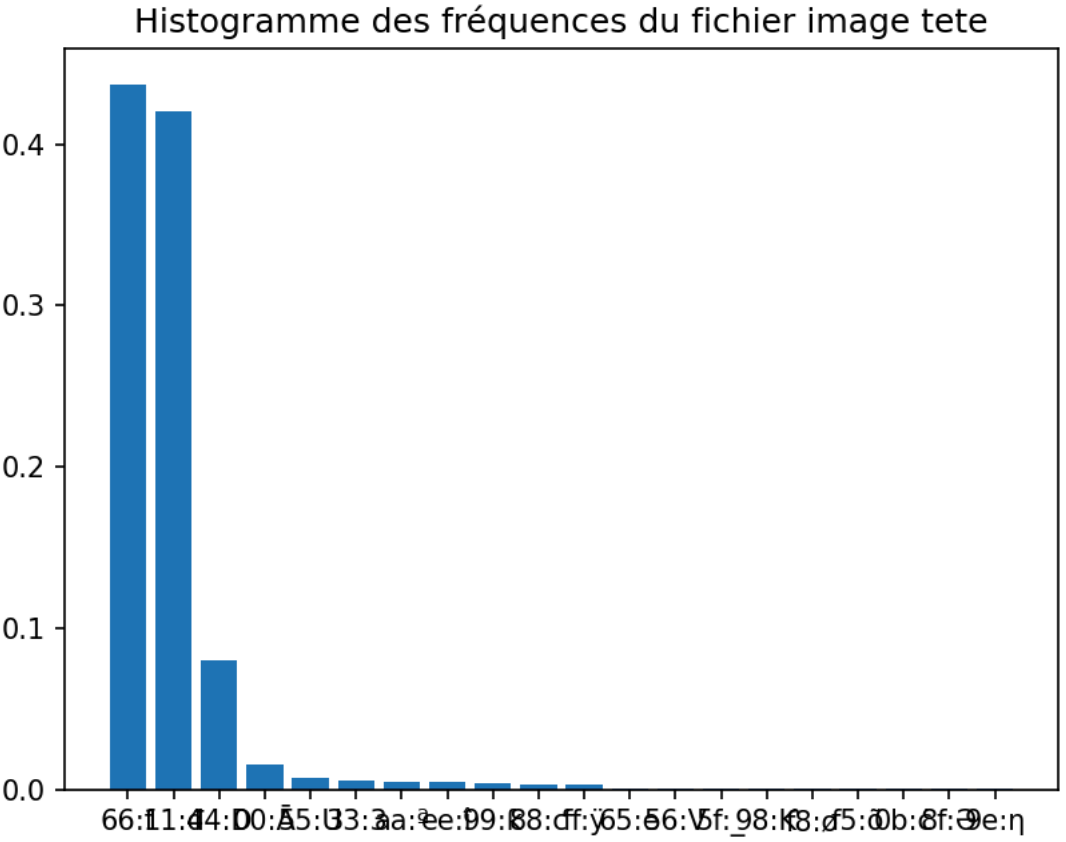

Histogramme des fréquences du fichier texte RougeNoir
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000



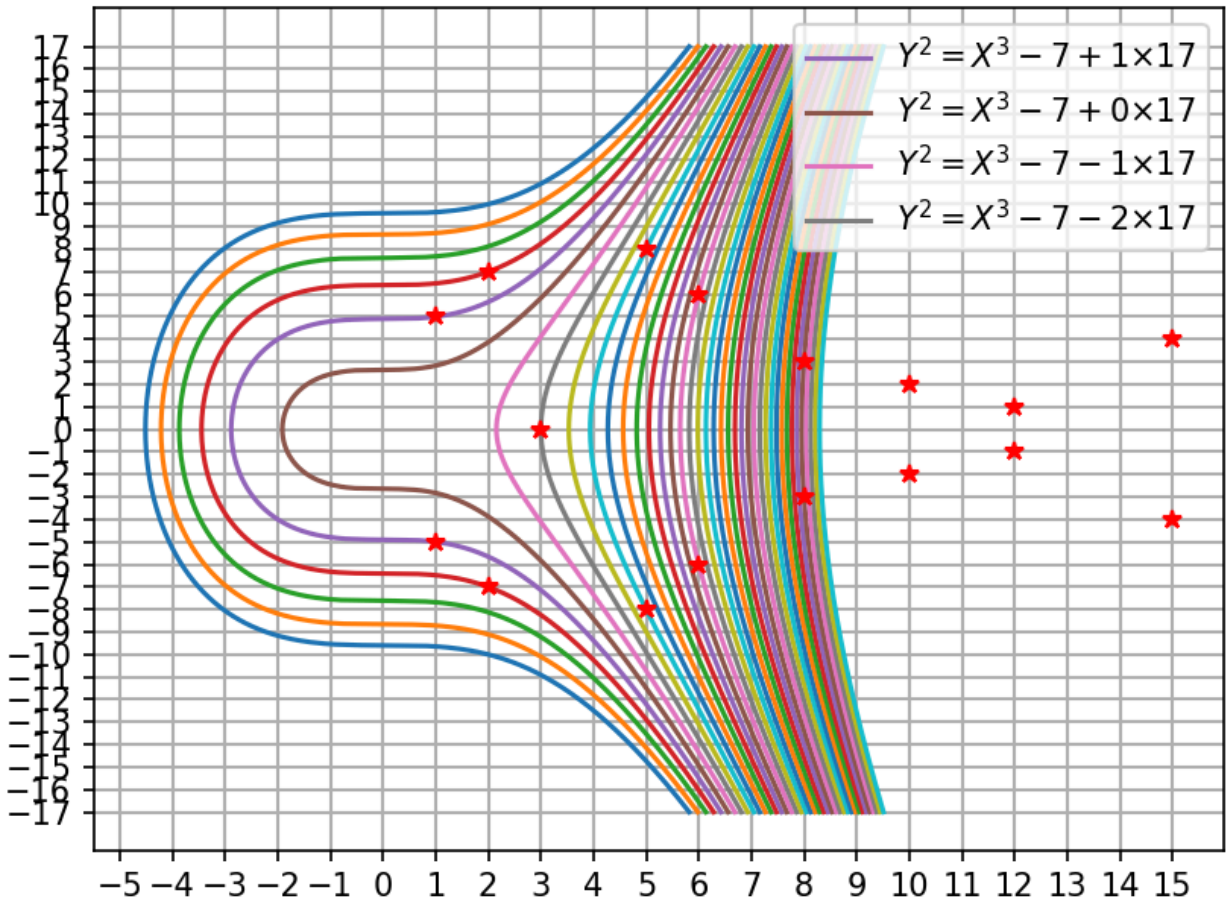
HISTOGRAMMES FICHIERS IMAGES :



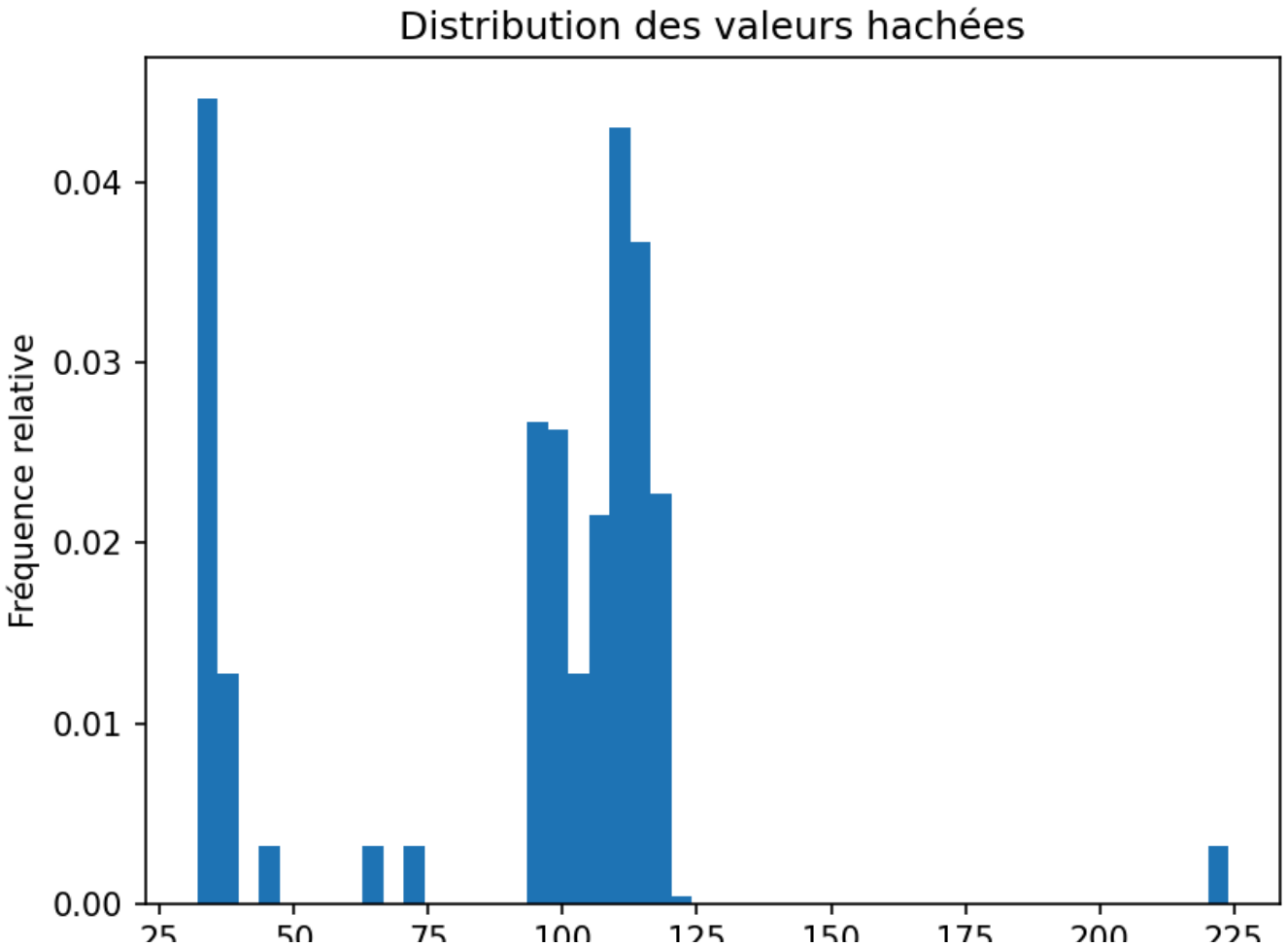

Histogramme des fréquences du fichier image coulb
0.6
0.5
0.4
0.3
0.2
0.1
0.0


Histogramme des fréquences du fichier image tete
0.4
0.3
0.2
0.1
0.0

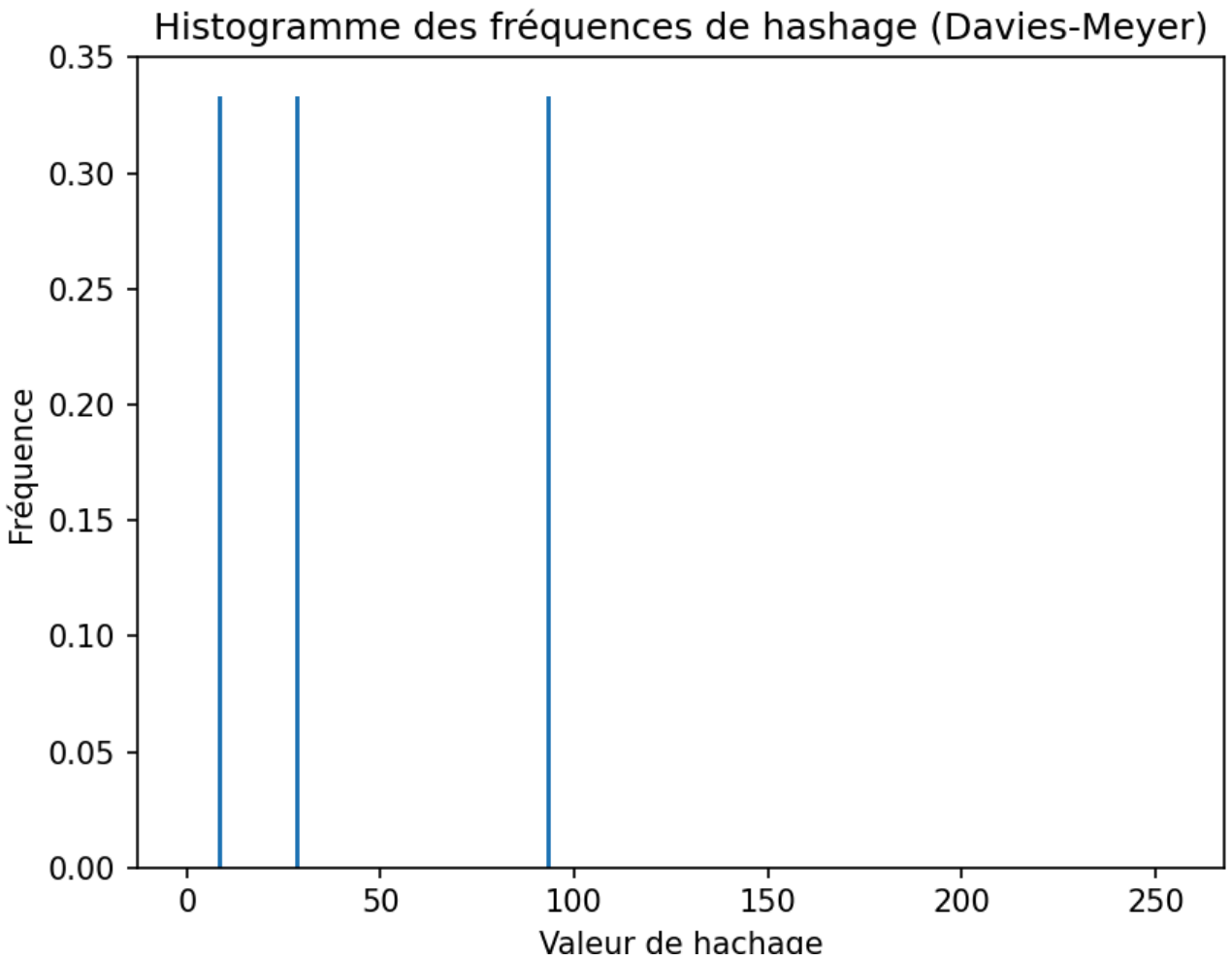
TP5 : Courbe elliptique :



TP7 : Hachage Davies Meyer



=> Hachage par DES



=> hachage naive