TP Python - Codes correcteurs d'erreurs

Contexte. Dans certains des schémas destinés à la cryptographie post-quantique, des erreurs sont volontairement introduites dans les messages chiffrés, que seul le propriétaire de la clé secrète pourra inverser. Nous allons parcourir pendant ce TD certains mécanismes de codes correcteurs d'erreur. Ces mécanismes de corrections d'erreurs sont également utilisés au quotidien, notamment dans les CD.

Exercice 1 - Code de répétition. Un code de répétition de longueur n fonctionne pour des messages élémentaires de 1 bit. L'encodage se fait en répétant ce bit n fois et le décodage en comptant le bit ayant l'occurence la plus élevée. Pour $n = 3$, on encode 0 en 000 et 1 en 111 ; les codes reçus décodés en 0 seront $001, 010, 100, 000$ et ceux décodés en 1 seront $110, 101, 011, 111$.

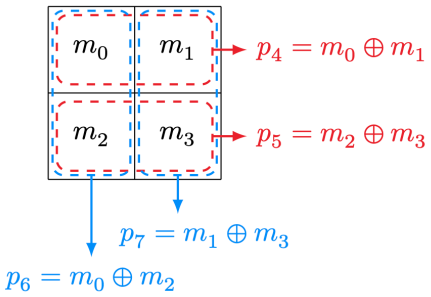
1. Étant donné un code de répétition de longueur n, combien de bits peuvent être modifiés sans que le décodage échoue ? Cette valeur, notée t est appelée capacité de correction du code.
2. Implémentez une fonction `encode_repeat(b, n)` qui prend en entrée un bit `b` et qui retourne l'entier correspondant à `n` répétitions du bit `b`.
3. Implémentez une fonction `decode_repeat(c, n)` qui décode le mot `c` sachant qu'il contient `n`.
4. Implémentez une fonction `channel(value, n)` qui modélisera un canal modifiant **aléatoirement** un bit du mot `value` et retournera le mot obtenu.
5. Écrivez une procédure de test pour ce premier code correcteur. Votre procédure effectuera une certaine quantité de fois les étapes suivantes:
   - Générer un bit aléatoire et une valeur n aléatoire comprise entre 3 et 100;
   - Le coder;
   - Le "transmettre" via le canal;
   - Le décoder;
   - Vérifier la conformité du décodage.

Exercice 2 - Code carré. Le facteur d'expansion d'un code est le rapport entre le nombre de bits transmis et le nombre de bits du message initial. Dans le cas précédent, le facteur d'expansion était au minimum de 3. Dans cet exercice, on s'intéresse à un code un peu plus avancé avec un facteur d'expansion égal à 2. Ce code a les paramètres suivants:

- Il prend en entrée des blocs de taille $k = 4$ bits. (dimension du code);
- Il retourne des mots de taille $n = 8$ bits. (longueur du code);

- Il permet de décoder une erreur de manière certaine.

Écrivons le message de 4 bits à encoder sous la forme $m_0m_1m_2m_3$. Le code carré leur concatène $n - k = 8 - 4 = 4$ nouveaux bits $p_4p_5p_6p_7$ (appelés **bits de parité** ou **bits de redondance**) de la manière suivante:



Ces relations entre les bits de parité p_j et les bits de message m_i sont appelées **équations de parité**.

1. Implémentez une fonction `encode_square(value)`, qui prend en entrée un message `value` de 4 bits représenté par un entier compris entre 0 et $2^4- = 15$ et encode ce message en un mot de 8 bits représenté par un entier compris entre 0 et 255.

2. Supposons maintenant qu'une erreur altère le mot de code. Si cette erreur se trouve parmi les quatre bits de message m_i, alors **deux** équations de parité ne seront plus vérifiées. L'identification de ces deux équations permet de retrouver le bit erroné.

   Si c'est un bit de parité p_j qui est altéré, alors une unique équation de parité sera non-satisfaite. Mais dans ce cas, les bits de message ne sont pas altérés, donc il n'y a rien à faire, si ce n'est retourner le bloc $m_0m_1m_2m_3$.

   Implémentez une fonction `decode_square(value)`, qui prend en entrée `value`, un bloc de 8 bits représenté par un entier compris entre 0 et 255, et décode ce bloc en un bloc de 4 bits représenté par un entier compris entre 0 et 15.

3. Testez la validité de vos fonctions en utilisant le canal modélise dans l'exercice précédent.

4. On souhaite maintenant encoder et décoder du texte, sous forme de chaînes de caractères. Pour cela, il faut découper nos chaînes en blocs de 4 bits, puis encoder successivement chacun des blocs obtenus.

   • Implémentez une fonction `cut(text)`, qui prend en entrée une chaine de caractères `text`, et qui retourne un tableau d'entiers compris entre 0 et 15 représentés sur 4 bits, dont la concaténation des bits est égale à celle de la chaîne de caractères.

   • Implémentez une fonction glue(tab), qui effectue de procédé inverse de la focntion cut(text).

• Implémentez une fonction `encode_message(text)`, qui prend en entrée une chaine de caractères `text`, et qui retourne un tableau d'octets (type python : `bytearray`) correspondant à la concaténation des encodages de blocs de 4 bits issus du découpage de `text` par la fonction `cut`.

• Implémentez une fonction `decode_message(tab)`, qui décode un tableau d'octets octet par octet, grâce à la fonction `decode_square` implémentée plus haut.

5. Implémentez une fonction `channel_message(tab)`, qui introduit une erreur aléatoire dans chaque octet du tableau d'octets `tab` passé en paramètre. Puis, encodez, altérez et décodez les chaînes de caractères de votre choix.

Exercice 3 - Comparaison des méthodes sur le canal binaire symétrique. Le canal utilisé pour les deux premiers exercices n'est pas très réaliste. En effet, il provoque **exactement** une erreur sur une position aléatoire du mot à transmettre, peu importe la longueur de ce mot.

Un canal plus réaliste est le **canal binaire symétrique** (souvent abrégé en BSC pour *binary symmetric channel*). Ce canal transmet le message bit par bit, et modifie chacun des bits de manière indépendante et avec probabilité p, où p est un paramètre compris entre 0 et 1.

1. Implémentez une fonction `bsc(block, p, size)` qui transmet un message `block` de longueur `size` via le canal binaire symétrique.

2. On souhaite mesurer le taux de transmission du canal, sur des messages de longueur 4 bits. Pour cela, on identifie 3 méthodes :

   (A) on transmet simplement le message à travers le canal, sans rien encoder ;

   (B) on encode chacun des 4 bits avec le code de répétition de longueur 3, puis on transmet les 12 bits correspondants à travers le canal, et on essaie de décoder le résultat ;

   (C) on encode un bloc de 4 bits avec le code carré, puis on transmet les 8 bits correspondants à travers le canal, et on essaie de décoder.

   Pour chacune des méthodes, écrire une fonction qui prend en paramètre p, effectue 1000 tirages aléatoires de messages de longueur 4, et compte le nombre de messages bien transmis à travers le canal.

3. En faisant varier p par pas de 0.001 entre 0 et 0.5, afficher dans un graphique (avec `matplotlib` ou `gnuplot` par exemple) le taux de transmission de chaque méthode en fonction de p. Quelle méthode transmet le mieux l'information ?

Exercice Bonus. Renseignez-vous et incorporez le **Code de Hamming** de dimension 4 et de longueur 7 aux comparatifs de l'exercice précédent, en précisant son facteur d'expansion.