Compte-rendu TP3 : Code de Hamming

Lyes SEHILA
L3 Informatique

15 février 2025

Table des matières

1 Rappel du sujet

Le TP avait pour objectif la conception et l'implémentation d'une application permettant :

— D'encoder un message binaire en utilisant le code de Hamming.

— De vérifier et corriger, le cas échéant, une trame binaire reçue.

L'application devait comporter une interface graphique (réalisée en Swing) ainsi qu'une interface en ligne de commande pour tester les différentes fonctionnalités du code de Hamming.

2 Analyse du sujet

L'analyse du sujet s'est focalisée sur deux grands axes :

1. **Méthode de calcul du code de Hamming** :
   — *Détermination des bits de contrôle* : Le programme calcule le nombre de bits de contrôle à ajouter en fonction de la longueur du message à encoder en respectant la relation $(2^i - 1) - i$.
   — *Placement des bits de contrôle* : Les bits de contrôle sont placés aux positions correspondant aux puissances de 2 (c'est-à-dire aux positions 1, 2, 4, 8, ...). Ce placement permet de détecter et de corriger les erreurs en cas de réception d'une trame erronée.
   — *Calcul des bits de contrôle* : Pour chaque bit de contrôle, on effectue une addition binaire (XOR) sur des positions déterminées par la règle de Hamming.
2. **Méthode de vérification et de correction** :
   — *Calcul du syndrome* :
      — Le **syndrome** est un ensemble de bits de contrôle recalculés à la réception. En code de Hamming, on recalcule chaque bit de contrôle en se basant sur la trame reçue.
      — Si le syndrome est **nul** (tous les bits à 0), alors on considère qu'aucune erreur n'est détectée.
      — Si le syndrome est **non nul**, il fournit la position (en binaire) du bit erroné.
   — *Correction de l'erreur* : En convertissant le syndrome en une position décimale, le programme identifie le bit erroné et l'inverse pour corriger la trame.

3 Choix techniques effectués

Les choix techniques majeurs pour ce projet sont :

— **Interface graphique moderne** : Utilisation de `Swing` pour une interface utilisateur claire, avec un thème sombre pour faciliter la lecture des résultats. Pour améliorer le design de l'interface, j'ai utilisé l'outil ChatGPT afin d'optimiser l'agencement et la présentation graphique.

— **Organisation du code** : Séparation de la logique (classe `HammingCode`) et de l'interface graphique (classe `HammingInterface`) pour une meilleure maintenabilité.

— **Algorithmes de calcul et de correction** : Mise en place des formules du code de Hamming pour le calcul des bits de contrôle et la détection/correction d'une éventuelle erreur.

4 Implémentation

Cette section détaille l'implémentation de l'algorithme de Hamming, depuis la génération des combinaisons binaires jusqu'à la correction des erreurs. L'approche adoptée s'appuie sur une structure modulaire en Java, garantissant clarté et maintenabilité.

4.1 Opérations binaires et génération de combinaisons

Pour réaliser l'addition binaire (opération XOR), la méthode `binaryAddition` compare deux bits et retourne 0 si les bits sont égaux, et 1 sinon. Les méthodes `generateBinaryCombination` et `binaryToDecimal` permettent de générer toutes les combinaisons possibles pour un bit dit « invariant » et de convertir un tableau binaire en valeur décimale. Ces étapes sont essentielles pour déterminer les indices utilisés lors du calcul des bits de contrôle.

4.2 Calcul des indices et placement des bits de contrôle

La méthode `calculateControlBitIndices` génère, pour chaque bit de contrôle, une liste d'indices correspondant aux positions des bits intervenant dans le calcul de la parité. Par ailleurs, `calculateControlBitsForEmission` détermine le nombre de bits de contrôle à ajouter selon la relation $(2^i - 1) - i$. Le placement des bits de contrôle dans le mot de Hamming est assuré par la méthode `determineControlBitPositions` qui positionne ces bits aux emplacements correspondant aux puissances de 2.

4.3 Encodage, vérification et correction

L'encodage s'effectue dans la méthode `encodeHamming` :

— Le message original est d'abord converti en liste d'entiers.

— Les bits de contrôle sont insérés aux positions définies et les bits de données sont placés dans les autres positions.

— Ensuite, le calcul de chaque bit de contrôle se fait par une opération XOR sur les indices déterminés précédemment.

Pour la vérification, la méthode `decodeHamming` recalcule les bits de contrôle à partir du message reçu afin de constituer le syndrome. Si ce dernier est non nul, la méthode `correctHamming` identifie la position erronée et corrige le message en inversant le bit concerné.

4.4 Extrait de code principal

Ci-dessous un extrait représentatif de la méthode d'encodage :

Listing 1 – Méthode d'encodage du code de Hamming

```
public static int[] encodeHamming(String message) {
    List<Integer> originalMessage = Arrays.asList(message.split("")).
       stream().map(Integer::parseInt).toList();
    int controlBits = calculateControlBitsForEmission(originalMessage.
       size());
    int[] encodedMessage = new int[controlBits + originalMessage.size()
       ];
    Arrays.fill(encodedMessage, 0);

    // Insertion des bits de contr le et des bits de donn es
    List<Integer> controlBitPositions = determineControlBitPositions(
       controlBits);
    controlBitPositions.sort(Comparator.reverseOrder());
    int globalIndex = encodedMessage.length - 1;
    int localIndex = 0;
    while (globalIndex >= 0) {
        if (!controlBitPositions.contains(globalIndex)) {
            encodedMessage[encodedMessage.length - 1 - globalIndex] =
               originalMessage.get(localIndex);
            localIndex++;
        }
        globalIndex--;
    }

    // Calcul des bits de contr le
    List<List<Integer>> controlBitIndices = calculateControlBitIndices(
       controlBits);
    for (int i = 0; i < controlBits; i++) {
        int binarySum = 0;
        for (int j = 0; j < controlBitIndices.get(i).size(); j++) {
            binarySum = binaryAddition(binarySum, encodedMessage[
               encodedMessage.length - controlBitIndices.get(i).get(j)])
               ;
        }
        encodedMessage[encodedMessage.length - 1 - controlBitPositions.
           get(i)] = (binarySum == 0) ? 0 : 1;
    }

    return encodedMessage;
}
```

Les méthodes de vérification et de correction suivent une logique similaire, assurant que le syndrome calculé permet de localiser et de corriger l'erreur éventuelle.

5 Résultats et tests

5.1 Exemple d'encodage

La Figure 1 montre l'encodage du message binaire `1011`. On y observe le message original et la trame encodée sous forme de code de Hamming. Les bits de contrôle C0, C1, C2 (positions de puissance de 2) ont été correctement calculés et insérés.

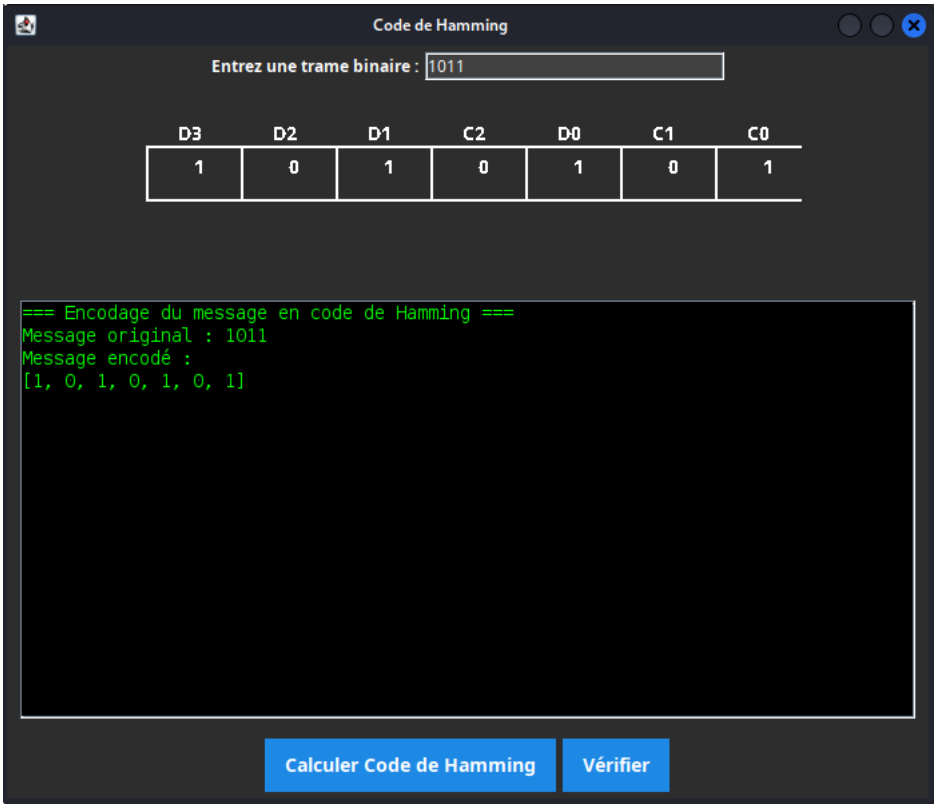

FIGURE 1 – Encodage du message 1011.

5.2 Exemple de vérification et correction

La Figure 2 illustre la vérification et la correction pour la trame reçue `1101101`. Après calcul des bits de contrôle, le programme détecte une erreur à la position 7 (représentée en binaire par `111`). Le bit erroné est alors inversé pour obtenir la trame corrigée `0101101`.

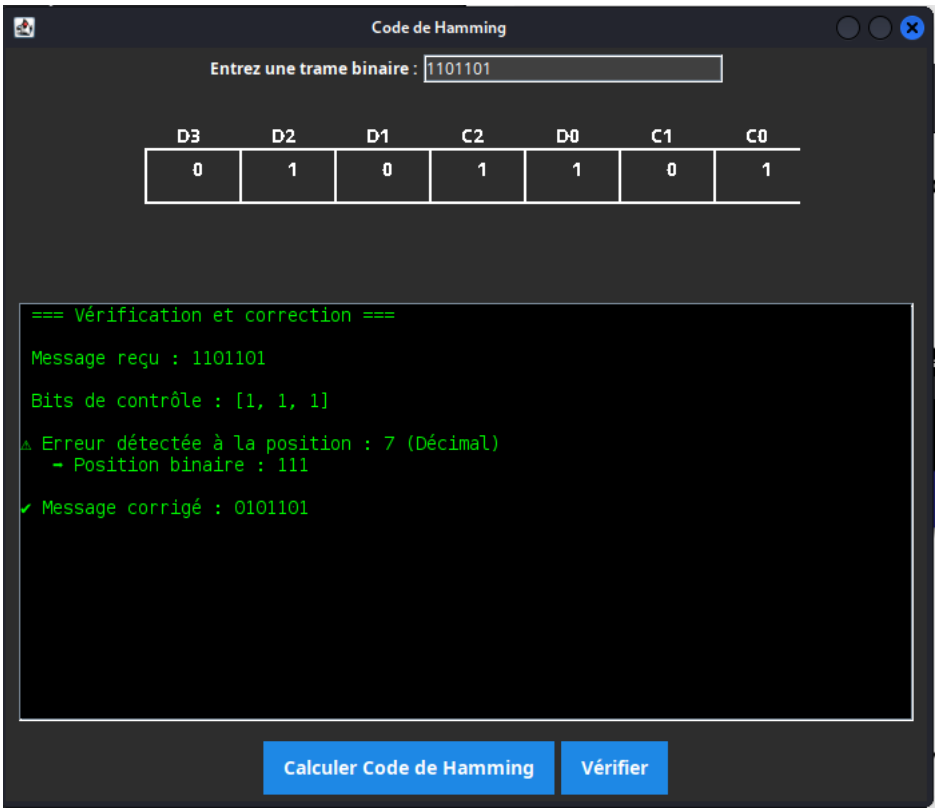

FIGURE 2 – Vérification et correction de la trame 1101101.

6 Difficultés et Perspectives

Le développement de cette application a permis de consolider les notions d'algorithmes de correction d'erreurs et de programmation orientée objet en Java. Parmi les points positifs et les défis rencontrés, on peut noter :

- **Difficultés rencontrées** :
  - La mise en œuvre du placement des bits de contrôle aux positions de puissances de 2.
  - L'intégration de la logique d'encodage/décodage dans une interface graphique conviviale et réactive.
- **Limites du programme** :
  - L'algorithme est actuellement conçu pour un format de message particulier. Une extension pour gérer des messages de longueur plus importante ou la détection de plusieurs erreurs serait envisageable.
  - L'interface graphique peut être enrichie pour proposer plus d'informations pédagogiques ou d'animations.

— **Perspectives d'évolution** :
  — Optimiser les algorithmes pour des messages de grande taille.
  — Améliorer l'interface graphique avec des visualisations interactives du calcul des bits de contrôle.

7 Conclusion

En conclusion, ce TP a offert une excellente opportunité de mettre en pratique des concepts théoriques de correction d'erreurs dans un cadre concret. Les fonctionnalités de base (encodage, vérification et correction) ont été mises en place avec succès, et plusieurs pistes d'amélioration sont envisageables pour enrichir et optimiser l'application à l'avenir.