Codes correcteurs d'erreurs

Projet : implémentation d'un LDPC

La but de ce projet est d'implémenter un code LDPC en Java et d'analyser ses performances. Une archive (`LDPC-starter.zip`) contenant une classe représentant des matrices binaires vous est fournie et sera à compléter tout au long du projet. 4 séances de 1h20 seront dédiées à ce projet. Ce projet sera **noté** et comptera pour la moitié de la note finale du module. Vous pouvez travailler en binôme. Vous devez rendre une archive contenant **votre code** (qui doit compiler) et **un rapport** expliquant vos choix et l'interprétation de vos résultats. De façon alternative, vous pouvez rendre une vidéo (contenant l'explication de votre code ainsi qu'une démonstration de son exécution) à la place du rapport.

Contexte

Vous êtes membre de l'équipage d'une base spatiale située sur une planète lointaine. Cependant, l'expédition tourne à la catastrophe : certains membres de l'équipage disparaissent dans des circonstances mystérieuses. Comme un malheur n'arrive jamais seul, l'équipement de la base s'avère vétuste, et vos coéquipiers doivent constamment se séparer pour effectuer diverse réparations. C'est notamment le cas du système de communications : celui-ci se dérègle régulièrement sans qu'on ne sache trop pourquoi, rendant les transmissions difficiles à exploiter. La dernière fois que cela est arrivé, vous étiez avec la docteure Blue, qui a décidé d'aller seule réparer la station de transmission. Ne la voyant pas revenir, et ne pouvant pas utiliser les caméras de surveillance pour vérifier que tout se passe bien (le signal était trop altéré pour cela), vous avez décidé d'aller vous-même effectuer les réparations. Sur le chemin, vous avez croisé le lieutenant Pink, qui semblait d'ailleurs anormalement nerveux. À peine les communications rétablies, vous avez reçu un signal d'alerte émis par Pink. celui-ci a retrouvé le corps de Blue à deux pas de la salle des communications, il dit vous avoir vu rôder près des lieux du crime quelques instants auparavant. Depuis, tout l'équipage vous soupçonne.

Il est temps de régler le problème des communications une bonne fois pour toutes. Vous vous portez volontaire, espérant par ailleurs redorer votre réputation. Vous proposez d'utiliser un code correcteur afin de permettre l'utilisation du système de communications malgré les altérations causées par les dérèglements de la station. Par chance, vous avez été formé à l'INSA CVL, vous êtes donc tout à fait compétent dans ce domaine. De par leur excellente capacité de correction s'approchant de la borne de Shannon, les codes LDPC vous semblent être le choix le plus judicieux.

Après analyse, lorsque la station est déréglée, le taux d'erreur est de 2% des bits transmis. La transmission étant assez lente, le rendement du code doit être d'au moins de 2/3 pour éviter les latences. De plus, l'information est transmise par mots de 4096 bits. Vous devez donc implémenter un code de paramètres $k = 4096$ et $n = 6144$, capable de détecter et corriger parfaitement des mots contenant au plus 2% de bits erronés (on s'autorisera un taux d'échec inférieur à $0,1\%$). Par précaution, on souhaite que les performances de ce code restent acceptables jusqu'à un taux d'erreur critique de $2,5\%$. Dans ce cas, on s'autorise un taux d'échec de décodage inférieur à 15%. En cas d'échec, le mot est renvoyé par l'expéditeur. Cependant, le code doit impérativement détecter tout mot erroné, et ne doit jamais décoder un mot qui n'est pas le bon.

Matériel

Avant de démarrer le TD, nous allons étudier le contenu de l'archive fournie.

Exercice 1. Ouvrez les fichiers `Matrix.java` et `Main.java` et étudiez leur contenu. Compilez et exécutez `Main.java`.

Le fichier `Matrix.java` contient le code de la classe `Matrix`, qui permet de manipuler des matrices binaires. Les attributs de cette classe sont deux entiers `rows` et `cols` correspondant respectivement au nombre de lignes et de colonnes d'une matrice, et un tableau de `byte`s nommé `data` contenant ses coefficients. La classe possède un constructeur prenant en paramètre un nombre de lignes et de colonnes et créant une matrice nulle aux dimensions correspondantes, et un autre constructeur créant une matrice à partir d'un tableau de `byte`s. En plus des accesseurs et mutateurs, cette classe possède les méthodes suivantes :

`boolean isEqualTo(Matrix m)` : Compare si deux matrices sont égales.

`void shiftRow(int a, int b)` : Permute la ligne d'indice `a` avec la ligne d'indice `b`.

`void shiftCol(int a, int b)` : Permute la colonne d'indice `a` avec la colonne d'indice `b`.

`void display()` : Affiche la matrice sur le terminal.

`Matrix transpose()` : Renvoie la transposée de la matrice.

`Matrix add(Matrix m)` : Additionne deux matrices et renvoie le résultat.

`Matrix multiply(Matrix m)` : Multiplie deux matrices et renvoie le résultat.

Par exemple, si on considère deux matrices A et B et qu'on souhaite afficher sur le terminal leur multiplication, on utilisera l'instruction `A.mult(B).display();`

Attention, une opération entre des variables de types `int` et `byte` renvoie une valeur `int`, il faudra donc prendre garde à caster correctement les opérations sur les coefficients de vos matrices.

Exercice 2. Créez quelques matrices et testez les fonctions d'addition, de multiplication et de transposition de la classe `Matrix`. Affichez le résultat.

Le fichier `Main.java` contient une méthode statique `Matrix loadMatrix(String file, int r, int c)` qui permet de charger une matrice depuis un fichier. La méthode prend en paramètres le nom du fichier, et le nombre de lignes et de colonnes de la matrice stockée dans le fichier. Le répertoire `data` contient deux fichiers contenant les matrices de contrôle de deux codes LDPC réguliers :

- `matrix-15-20-3-4` contient une matrice de 15 lignes de poids 4 et de 20 colonnes de poids 3.
- `Matrix-2048-6144-5-15` contient une matrice de 2048 lignes de poids 15 et de 6144 colonnes de poids 5.

Par exemple, pour charger la matrice du deuxième fichier dans une variable H, on utilisera l'instruction : `Matrix H = loadMatrix("data/Matrix-2048-6144-5-15", 2048, 6144);`

Les deux matrices qui vous sont fournies ont des propriétés intéressantes. Pour chacune d'elles, en notant H la matrice, r le nombre de lignes et c le nombre de colonnes, et en posant $H = (L|R)$ où L est une matrice $r \times n - r$ et R est une matrice carrée $r \times r$, on a que R est inversible. Cela signifie d'une part que H est de rang plein et donc que le code associé a pour paramètres $k = c - r$ et $n = c$, et d'autre part que le code correspondant est systématique. De plus, la maille du graphe de Tanner associé à H est strictement supérieure à 4 (le graphe ne contient donc pas de cycles courts).

Première tâche : l'encodage

La première tâche consiste à implémenter la fonction d'encodage de notre code LDPC. Afin de tester les différentes étapes de cette tâche, on utilisera la matrice stockée dans le fichier `matrix-15-20-3-4`.

La classe matrice possède déjà des méthodes pour permuter les lignes et colonnes d'une matrice. Afin d'utiliser la méthode du pivot de Gauss le plus simplement possible, nous aurons besoin d'une méthode permettant d'additionner une ligne d'une matrice à une autre ligne.

Exercice 3. Ajoutez les méthodes `public void addRow(int a, int b)` et `public void addCol(int a, int b)` à la classe `Matrix`. La méthode `addRow` (resp. `addCol`) ajoute la ligne (resp. colonne) d'indice `a` à la ligne (resp. colonne) d'indice `b` de la matrice sur laquelle elle s'applique. Testez vos méthodes sur la matrice du fichier `matrix-15-20-3-4`.

Les matrices qui sont fournies sont de la forme $H = (L|R)$ où R est inversible. On peut donc, en utilisant le pivot de Gauss sur les lignes de H, transformer H en une matrice $H' = (M|\mathsf{id})$ (où id est l'identité) telle que H et H' sont des matrices de contrôle du même code linéaire C. Par abus de langage, on appellera H' la matrice de contrôle de C *sous forme systématique*.

Exercice 4. Ajoutez la méthode `public Matrix sysTransform()` à la classe `Matrix`. Cette méthode s'applique sur une matrice de contrôle H d'un code C qu'on supposera de la forme $H = (L|R)$ où R est inversible, et renvoie la matrice de contrôle de C sous forme systématique. Testez cette méthode sur la matrice du fichier `matrix-15-20-3-4`.

Il est très facile de déterminer la matrice génératrice sous forme systématique d'un code depuis sa matrice de contrôle lorsqu'elle est sous la forme $H' = (M|\mathsf{id})$: il suffit de calculer la matrice $G = (\mathsf{id}|M^t)$ (attention, la notation id désigne la matrice identité et dépend du contexte dans lequel elle est utilisée, en particulier la dimension de id n'est pas forcément la même dans G et H).

Exercice 5. Ajoutez la méthode `public Matrix genG()` à la classe `Matrix`. Cette méthode s'applique sur une matrice de contrôle H' d'un code C qu'on supposera de la forme $H' = (M|\mathsf{id})$, et renvoie la matrice $G = (\mathsf{id}|M^t)$ génératrice de C sous forme systématique. Testez cette méthode sur la matrice du fichier `matrix-15-20-3-4` (il faudra au préalable mettre cette matrice sous la forme adéquate).

Une fois qu'on connaît la matrice génératrice d'un code linéaire G sous forme systématique, l'encodage d'un mot u revient simplement à multiplier u par G. l'encodage x de u est donc $x = u \cdot G$.

Exercice 6. En utilisant une méthode de la classe `Matrix`, encodez le mot `u=10101` avec le code systématique admettant la matrice du fichier `matrix-15-20-3-4` pour matrice de contrôle. On notera x la variable contenant l'encodage de `u`.

Deuxième tâche : le décodage

Le décodage des codes LDPC se fait via une représentation de la matrice de contrôle sous forme d'un graphe de Tanner. Nous allons créer une nouvelle classe `TGraph` permettant de représenter ces graphes. Considérons une matrice H de `n_r` lignes de poids `w_r` et `n_c` colonnes de poids `w_c`. Concrètement, nous représenterons son graphe de Tanner par les deux tableaux de dimension 2 suivants :

— `left`, un tableau de `n_r` lignes et `w_r + 1` colonnes où chaque ligne correspond à une ligne de la matrice tel que pour tout $j > 0$, `left[i][j]` est l'indice du j-ème bit à 1 dans la i-ème ligne de H, et

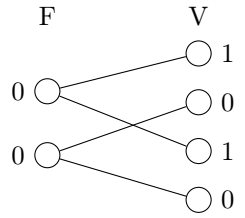
— `right`, un tableau de `n_c` lignes et `w_c + 1` colonnes tel que pour tout $j > 0$, `right[i][j]` est l'indice du j-ème bit à 1 dans la i-ème colonne de H.

On fait correspondre chaque nœud variable (resp. fonctionnel) du graphe de Tanner à une ligne de `right` (resp. `left`). Ainsi, pour chaque ligne i correspondant à un nœud N dans le graphe de Tanner, les valeurs `right[i][j]` (resp. `left[i][j]`) pour $j > 1$ sont les les indices des lignes de `left` (resp. `right`) qui correspondent aux nœuds voisins de N. On donnera à `right[i][0]` (resp. `left[i][0]`) la valeur du bit associé à N.

Par exemple, considérons la matrice :

$$H = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{pmatrix}.$$

En associant aux nœuds variables du graphe de Tanner de H les bits du mot `1010`, on obtient le graphe :



Dans la classe `TGraph`, ce graphe sera représenté par les deux tables suivantes :

`left` :

0	**0**	**2**
0	**1**	**3**

`right` :

1	**0**
0	**1**
1	**0**
0	**1**

Créez un fichier `TGraph.java` contenant la classe `TGraph`. Cette classe prendra 4 attributs `n_r`, `w_r`, `n_c` et `w_c` de type `int` et 2 attributs `left` et `right` de types `int[][]`. Implémentez le constructeur `public TGraph(Matrix H, int wc, int wr)` (on supposera que ce constructeur sera toujours appelé avec une matrice régulière dont les lignes (resp. colonnes) sont de poids `wr` (resp. `wc`). Ce constructeur devra instancier un `TGraph` à partir de la matrice de contrôle H passée en argument. On instanciera les bits associés aux nœuds du graphe la valeur 0.

Pour tester que tout fonctionne bien, il serait commode de pouvoir afficher nos graphes de Tanner.

Exercice 7. Ajoutez une méthode `public void display()` à la classe `TGraph`. Cette méthode devra afficher le contenu des deux tableaux `left` et `right`. Créez un objet `TGraph` représentant le graphe de Tanner de la matrice du fichier `matrix-15-20-3-4` et affichez-le avec votre méthode `display()`.

Nous avons maintenant tout ce qu'il nous faut pour implémenter l'algorithme de correction par renversement de bits ! Pour cela, je vous conseille d'utiliser le pseudo-code que nous avons étudié en cours.

Exercice 8. Ajoutez une méthode `public Matrix decode(Matrix code, int rounds)` à la classe `TGraph`. Cette méthode devra effectuer `rounds` itérations de l'algorithme de correction par renversement de bits sur le mot `code` (on supposera que `code` est de la bonne longueur). La méthode devra renvoyer le mot corrigé. Si l'algorithme échoue, on renverra un vecteur de taille n dont tous les coefficients ont la valeur `-1` (où n désigne la longueur du code).

Il ne nous reste plus qu'à tester tout cela.

Exercice 9. On fixe le nombre d'itérations de l'algorithme de décodage à `round=100`. Corrigez le mot `x` calculé à la fin de la partie sur l'encodage, affichez le résultat. Créez les vecteurs d'erreurs de poids 2 suivants :

$$e_1 = (0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0),$$
$$e_2 = (0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0),$$
$$e_3 = (0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0),$$
$$e_4 = (0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0).$$

Pour chaque e_i, créez le mot $y_i = x + e_i$, calculez et affichez le syndrome de y_i, puis utilisez votre algorithme de décodage sur y_i, et comparez le résultat avec x. Quels sont les mots correctement corrigés ? Quels sont les mots pour lesquels la correction est erronée ? Quels sont les mots pour lesquels la correction échoue ?

Troisième tâche : l'évaluation

Maintenant que tout fonctionne correctement, nous pouvons utiliser nos fonctions d'encodage et de décodage sur des codes plus grands, et donc plus performants. Dans cette partie, nous allons analyser les performances du code LDPC défini par la matrice de contrôle stockée dans le fichier `Matrix-2048-6144-5-15`.

Exercice 10. Le fichier `Matrix-2048-6144-5-15` contient une matrice H régulière de rang plein et de dimensions 2048×6144. Le poids des lignes de H est de 15 et le poids de ses colonnes est de 5. Donnez le longueur des mots sources k, la longueur des mots de code n, la redondance r et le rendement τ du code LDPC admettant H pour matrice de contrôle. En supposant que les mots de code transitent sur un canal binaire symétrique dont la probabilité d'erreur est $p = 2 \cdot 10^{-2}$, combien d'erreurs en moyenne contiendront les mots reçus ? Même question pour $p = 2,5 \cdot 10^{-2}$.

Pour commencer, nous allons encoder un mot arbitraire avec ce code.

Exercice 11. À l'aide de vos méthodes,calculez la matrice génératrice systématique G du code défini par la matrice de contrôle H du fichier `Matrix-2048-6144-5-15`. Générez un mot u dont les coefficients sont des 1 aux indices pairs et des 0 sinon. Encodez ce mot à l'aide de G. On notera x le mot de code obtenu. Générez le graphe de Tanner associé à la matrice H.

Nous allons analyser le taux de réussite, le taux d'échec et le taux d'erreur de décodage de notre code LDPC pour des mots contenant un nombre w fixé d'erreurs. Comme c'est un code linéaire, pour générer un mot contenant w erreurs, il suffit de lui ajouter un mot de poids w. Nous avons donc besoin d'une méthode générant des mots de poids choisis.

Exercice 12. Ajoutez une méthode `public Matrix errGen(int w)` à la classe `Matrix`. Cette méthode devra générer aléatoirement un vecteur ligne de poids w. Pour cela, on pourra utiliser une boucle de w itérations : à chaque itération, on choisit aléatoirement l'indice d'un coefficient du vecteur. Si le coefficient est 0, on le remplace par 1, sinon on rajoute une itération à la boucle (pour générer un entier aléatoire en Java, utilisez un objet de la classe `Random` et la fonction `nextInt`).

Il ne nous reste plus qu'à déterminer empiriquement le taux de réussite, le taux d'échec et le taux d'erreur de décodage de notre code LDPC pour un nombre d'erreurs que nous ferons varier entre 124 et 154. Cette analyse nous permettra de conclure si ce code atteint des performances suffisantes pour notre application.

Exercice 13. Dans une boucle de 10^4 itérations, générez à chaque itération un mot e de poids $w = 124$, calculez le mot $y = x + e$, décodez ce mot en utilisant la méthode `decode` sur `round=200` itérations et comparez le résultat avec x. Déterminez si le décodage a réussi, échoué, ou s'il a renvoyé un mot erroné. À la fin des 10^4 itérations, donnez le pourcentage de réussite, d'échec et d'erreurs de décodage pour $w = 124$. Recommencez l'expérience pour $w = 134$, $w = 144$ et $w = 154$. Déduisez-en si les performances de ce code sont suffisantes pour l'application décrite dans ce TD.

Finalement, on peut se demander s'il aurait été possible d'obtenir des performances de correction similaires en utilisant un code avec un meilleur rendement. Pour cela, nous devons utiliser le théorème de Shannon.

Exercice 14. Donnez une borne théorique au rendement des codes correcteurs lorsqu'ils sont utilisés sur un canal binaire symétrique dont la probabilité d'erreur est $p = 2 \cdot 10^{-2}$. Que faudrait-il faire pour améliorer les performances de notre code LDPC ?