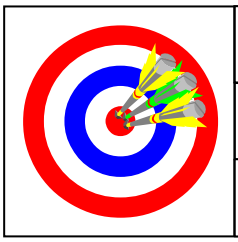


Algorithmes avancés

Codes correcteurs d'erreurs

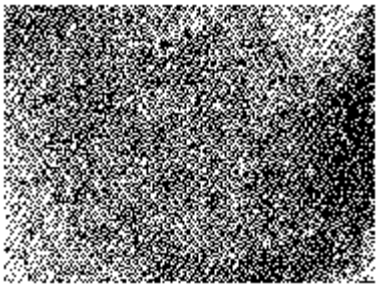
ITI - orientation logiciel & systèmes complexes — Travaux pratiques
hepia, HES-SO // Genève

Objectifs

- Etudier et implémenter les codes de Reed-Muller $RM(1,r)$ du 1er ordre de longueur r
- Pouvoir encoder un mot de $(r+1)$-bits en un mot de 2^r-bits en utilisant un code $RM(1,r)$ ainsi que décoder ce dernier pour récupérer le mot original
- Pouvoir bruiter un mot du code $RM(1,r)$ et débruiter ce dernier pour récupérer le mot encodé
- Lire ou écrire une image dans un fichier, pouvoir encoder/décoder cette image, ainsi que la bruiter/débruiter

Enoncé

Lors de la transmission des premières images de Mars, le brouillage du message durant son voyage intersidéral était tel que l'image reçue sur Terre ressemblait à cela :



Cette photographie a été prise en 64 niveaux de gris. Chaque pixel occupait donc initialement un mot de 6 bits. Afin de préserver l'intégrité des données lors de la transmission, chaque pixel a été codé sur un mot de 32 bits, ce qui permet de créer une redondance d'information et, s'il n'y a pas trop d'erreurs à la réception, de les corriger.

Il s'agit donc de mettre en œuvre des algorithmes de codage, de débruitage et de décodage qui permettent de retrouver les mots de départ (sur 6 bits) à partir des mots codés et brouillés qu'on a reçus (sur 32 bits).

Travail à réaliser

- Après avoir bien étudié les codes de Reed-Muller *RM*(1,*r*) du 1er ordre de longueur *r*, les implémenter pour une valeur arbitraire de *r* et les tester.
- Implémenter dans un paquetage le code de Reed-Muller *RM*(1,*r*)
  - Utiliser la classe `BigInteger` pour stocker les mots sous forme d'entiers.
  - Ecrire et tester les méthodes de `codage()` et de `decodage()`.
  - Implémenter les algorithmes de débruitage par :
    - recherche exhaustive sur la moitié des éléments du code (recherche semi-exhaustive) ;
    - recherche rapide.
- Ecrire une application console qu'on pourra lancer en ligne de commande. Cette application aura deux modes de fonctionnement pour traiter :
  1. de simples mots (vu comme des entiers) ;
  2. des images au format PGM stockées dans des fichiers textes (vu comme une liste d'entiers, un pixel étant un entier).
- Ecrire une méthode `bruit()`, en dehors du paquetage du code de Reed-Muller, qui prend en paramètres un mot et un seuil de probabilité et retourne le mot bruité. Elle parcourt chaque bit du mot et l'inverse avec une probabilité donnée par ce seuil. Utiliser cette méthode pour bruiter une image encodée en l'appliquant à chacun des pixels encodés.
- Pouvoir combiner les opérations de codage, bruitage, débruitage, décodage que ce soit pour un mot ou pour une image. En particulier, tester la chaîne complète de transformation :

  donnée originale —*encodage*→ donnée encodée —*brouillage*→ donnée bruitée

  donnée bruitée —*débruitage*→ donnée encodée —*décodage*→ donnée originale
- Débruiter les image de Mars encodées et bruitées qui sont fournies, et les stocker au format PGM pour les visualiser.

Données

Dans le cours **Algorithmes avancés** sur **https://cyberlearn.hes-so.ch/course/**, vous trouverez des fichiers :

- **`Main_skel.java`** : ce fichier contient la définition des menus utilisateur et des formats de sortie des données pour l'application en mode console ; à noter que les impressions des menus sont envoyées sur le canal d'erreur (via `System.err.print()`) pour les différencier des sorties de l'application, lesquelles sont envoyées sur la sortie standard (via `System.out.print()`).
- Des images de Mars encodées et bruitées, dont les pixels sont stockés sous forme d'entiers (mots de 32 bits, les entiers étant habituellement stockés sur 32 bits).
- Une autre image pour vos tests.

Complément

Le format PGM est structuré de la manière suivante :

- la première ligne contient la chaîne de caractères "P2"
- les lignes de commentaires commencent par # et ne comptent pas
- la ligne suivante contient le nombre de colonnes et de lignes (2 entiers)
- la ligne d'après contient le niveau de gris le plus grand, ici 63 (1 entier).
- une suite d'entiers, ici entre 0 et 63 (les retour à la lignes n'ont pas d'importance), où chaque entier est le niveau de gris d'un pixel, qui parcourent l'image de gauche à droite et de haut en bas

Il est bien entendu qu'une fois codée, l'image n'est plus au format PGM dans le sens où les données stockées ne sont plus entre 0 et 63, et ne correspondent donc plus directement au niveau de gris d'un pixel.

Consignes et rendu

Vous devez placer l'ensemble des fichiers et paquetages de votre application dans un **dossier à votre nom** (seulement votre nom, pas le prénom)**, tout en minuscules sans espaces ni accents**. Ce dossier contiendra en particulier au plus haut niveau la classe principale **obligatoirement** appelée **`Main.java`**, basée sur le fichier Main_skel.java. Vous zipperez ce dossier dans une **archive .zip** du même nom (**p.ex. `feka.zip` pour l'étudiant Lucien Feka**) ; ceci peut être fait en ligne de commande avec `zip -r feka.zip feka/`).
Cette archive devra être déposée dans sur **https://cyberlearn.hes-so.ch/course/**,

Algorithmes avancés→ Travail pratique : codes de Reed-Muller

Une fois l'archive dézippée, la compilation de `Main.java` doit pouvoir se faire dans le dossier résultant simplement avec la commande **`javac Main.java`** et le lancement de l'application avec **`java Main`**

Attention ! Vous ne devez rien modifier ni aux arguments passés en ligne de commande, ni au format de sortie des données, tels que spécifiés dans le fichier `Main_skel.java`. Vous ne devez pas non plus ajouter d'autres entrées au clavier ou de sorties à l'écran. Le non respect des consignes sera sanctionné.

Vous avez l'équivalent de 3 séances pour réaliser ce TP.
La date de rendu sera indiquée dans l'agenda du cours.
Votre application doit être bien structurée.
Les méthodes doivent être testées et les codes sources commentés.