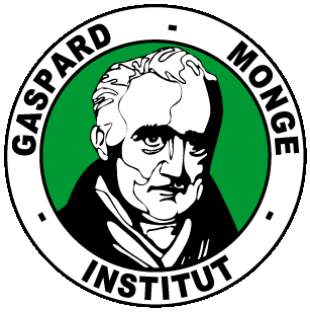


UNIVERSITÉ
PARIS-EST
MARNE-LA-VALLÉE

Polynômes irréductibles dans $\mathbb{F}_2[X]$, codes correcteurs BCH et QR codes

Kévin TRANCHO

Licence Informatique 3ème année

Projet Tuteuré encadré par Marc ZIPSTEIN et Jean-Yves THIBON

Année 2016-2017

Table des matières

Remerciements

Je tiens tout d'abord à remercier Monsieur **Marc Zipstein**, enseignant à l'université Paris-Est Marne-La-Vallée, aussi bien pour avoir accepté d'encadrer ce projet tuteuré sans que je m'y sois pris longtemps à l'avance, que pour sa disponibilité et son aide par des explications me permettant d'avancer et mener à bien ce projet.

J'aimerai aussi vivement remercier Monsieur **Jean-Yves Thibon**, enseignant à l'université Paris-Est Marne-La-Vallée, pour avoir suivi et encadré ce projet depuis le début mais aussi pour sa disponibilité et ses explications en particulier pour les parties plus théoriques.

Je veux aussi remercier **Victor Veillerette**, étudiant de ma promotion, pour avoir accepté de contribuer à la recherche de polynômes irreductibles en mettant à contribution son serveur personnel.

Chapitre 1

Présentation du projet

1.1 Objectifs

L'objectif de ce projet était de m'initier à la **cryptographie**, en continuité du cours de **Mathématiques pour l'informatique** enseigné par monsieur Thibon, ici par le travail avec des **polynômes irréductibles** dans un **corps fini**.
Le but était d'utiliser ces corps finis pour construire une famille de codes correcteurs : ici les **BCH** (Bose, Ray-Chaudhuri et Hocquenghem).

L'intérêt des codes correcteurs est de pouvoir envoyer un message et corriger ses erreurs et ça malgré une détérioration des données, ce qui est utilisé par exemple avec les DVD ou lors d'envois d'informations dans l'espace.
Nous nous sommes ensuite intéressés à la construction de **QR codes** qui utilisent les BCH et les codes de Reed-Solomon pour corriger les erreurs.

1.2 Organisation

Le projet a débuté en **Mars** pour se terminer en **Juin** et a été dès le début encadré par messieurs Zipstein et Thibon.
J'ai tout d'abord bénéficié d'une introduction aux corps finis et aux polynômes irréductibles dans $\mathbb{F}_2$ par messieurs Thibon et Zipstein.
Et pour commencer en autonomie monsieur Thibon m'a prêté son livre 'Error-Correcting Codes and Finite Fields' d'Oliver Pretzel et m'a dirigé sur la page Wikipédia des 'Corps finis'.
J'ai aussi par moments à l'improviste bénéficié de l'aide et des explications de monsieur Zipstein.
Pour la partie sur les BCH, monsieur Thibon a accepté de me faire un cours sur les BCH et codes de Reed-Solomon.
Ensuite, monsieur Zipstein m'a expliqué à travers des exemples de TD comment effectuer le codage et décodage des BCH.
Pour me lancer sur les QR codes, monsieur Thibon m'a partagé la spécification sur la norme des QR codes.
Pour finir j'ai fait deux démonstrations du programme à monsieur Zipstein, une pour la

partie polynômes et une autre pour les QR codes.

1.3 Outils de développement

Pour la réalisation du projet, nous avons choisi le **langage C**, dans le but d'implémenter intégralement chaque fonctionnalité que je pouvais utiliser.
En complément pour l'affichage et création des QR codes, j'ai utilisé la bibliothèque graphique **SDL**, simple à utiliser et adaptée à l'édition d'image dont je pouvais avoir besoin (édition des pixels, sauvegarde et changement d'images).

FIGURE 1.1 – *Logo de SDL*

1.4 Bilan général

Au moment de l'écriture du rapport, le projet est composé de 10 000 lignes de code source C.
C'est-à-dire plus de 400 fonctions (externes et locales confondues).
Le projet est séparé en 16 modules.
Pas de cas de crash connu référencé pour le moment.

1.5 Étapes de développement

Dans les sections suivantes vous trouverez le cheminement de mon raisonnement pour l'implémentation des différentes fonctionnalités et différentes parties :

- Création de polynômes dans $\mathbb{F}_2[X]$
- Opération sur les polynômes de $\mathbb{F}_2[X]$
- Recherche de polynômes irréductibles par crible d'Eratosthène
- Implémentation de l'Algorithme de Berlekamp et utilisation pour une recherche récursive de polynômes irréductibles à partir d'un Polynôme généré aléatoirement
- Construction des BCH : codage et décodage avec correction des erreurs
- Implémentation des polynômes de $\mathbb{F}_{2^q}[X]$ par construction de $\mathbb{F}_{2^q}$ à partir d'un polynôme irréductible de $\mathbb{F}_2[X]$
- Construction des QR codes : codage et décodage avec correction des erreurs

J'ai moi-même bénéficié d'un mémoire par Virginie FORICHON : 'CODES CORRECTEURS D'ERREURS : CONSTRUCTION ET EXEMPLES' datant de 2002-2003. Alors j'ai choisi d'expliquer mon raisonnement dans l'esprit de permettre au lecteur de comprendre au mieux les notions dont je traiterai et ceci aussi dans le cas d'une possible implémentation de QR codes par exemple.

Chapitre 2

Corps finis

2.1 Définition

Avant de définir un corps, donnons brièvement les propriétés de groupes et d'anneau sur lesquelles nous travaillerons.

Un groupe $(E, +)$ est défini tel que :
- Il existe 0 tel qu'il est l'élément neutre de l'opération $+$
  $\exists 0 \in E \quad \forall x \in E \quad x + 0 = 0 + x = x$
- $+$ est associative
  $\forall (x, y, z) \in E^3 \quad x + (y + z) = (x + y) + z$
- Pour tout $x \in E$, il existe x^{-1} tel que x^{-1} est l'inverse de x par $+$
  $\forall x \in E \quad \exists x^{-1} \in E \quad x + x^{-1} = 0$

Un groupe est commutatif si $\forall (x, y) \in E^2 \quad x + y = y + x$.

Un anneau $(E, +, \times)$ est défini de la manière suivante :
- $(E, +)$ est un groupe commutatif
- $\times$ possède un élément neutre que nous noterons 1
- $\times$ est associative
- $\times$ est distributive par rapport à $+$
  $\forall (x, y, z) \in E^3 \quad x \times (y + z) = x \times y + x \times z$
  $(y + z) \times x = y \times x + z \times x$

Si $\times$ est commutatif, alors l'anneau est commutatif.

On définira un corps $\mathbb{K} = (E, +, \times)$ de la manière suivante :

- $(\mathbb{K}, +, \times)$ est un anneau
- $(\mathbb{K}^*, \times)$ est un groupe, autrement dit on a aussi que tout élément de E admet un inverse par $\times$

2.2 Le corps de départ $\mathbb{F}_2$

Pour débuter notre travail sur les corps finis, nous commençons par manipuler $\mathbb{F}_2$, c'est-à-dire le corps fini formé par les entiers modulo 2, donc à valeur dans $\{0, 1\}$.

— On muni $\mathbb{F}_2$ de l'opération 'xor', c'est-à-dire le 'ou exclusif' en tant qu'addition et avec pour élément neutre 0.
— De plus on muni aussi $\mathbb{F}_2$ de la 'multiplication' classique avec pour élément neutre 1

Ce qui nous donne les tables suivantes :

+	0	1
0	0	1
1	1	0

×	0	1
0	0	0
1	0	1

On remarque aussi que le 'xor' est son propre inverse, en particulier dans $\mathbb{F}_2$

2.3 Calculs dans $\mathbb{F}_2[X]$

Pour l'implémentation des structures pour gérer ces polynômes j'ai choisi deux types de structures :
La première sous forme d'entier pour des opérations plus rapides mais limitées au degré inférieur ou égal à 31 (utilisée principalement dans le crible d'Eratosthène.
Une seconde extensible car allouée dynamiquement où chaque coefficient est stocké sur un bit.

2.3.1 Addition - Soustraction

En travaillant avec des polynômes de $\mathbb{F}_2[X]$, l'addition et la soustraction sont la même opération sur les coefficients, c'est-à-dire le 'xor'.
Exemple :

$$x^2 - x - 1 = x^2 + x + 1$$

De plus, l'addition devient elle-même sa propre opération inverse, et tout inverse d'un élément par l'addition est lui-même :

$$\forall x \in \mathbb{F}_2 \quad x + x = 0$$

On peut généraliser aux polynômes :

$$\forall p \in \mathbb{F}_2[X] \quad p + p = 0$$

2.3.2 Multiplication

La grande différence avec des polynômes de $\mathbb{Z}[X]$ pour la multiplication est qu'on calcule les coefficients modulo 2.
Exemple :

Dans $\mathbb{Z}[X]$	Dans $\mathbb{F}_2[X]$
$(x+1)(x+1)$	$(x+1)(x+1)$
$= x^2 + 2x + 1$	$= x^2 + 1$

Pour l'implémentation de la multiplication, j'ai choisi de la calculer comme la somme avec le 'xor' du premier terme multiplié à chaque coefficient non nul.
Ce qui nous fait simplement décaler le premier terme par le degré du coefficient en cours et effectuer l'addition à ce qui a déjà été calculé.
Exemple de mon choix de méthode de calcul informatique :

$$\begin{aligned}(x^2+1)(x^2+x+1) &= x^2(x^2+1) + x(x^2+1) + (x^2+1)\\ &= (x^4+x^2) + (x^3+x) + (x^2+1)\\ &= x^4 + x^3 + x + 1\end{aligned}$$

2.3.3 Division euclidienne

Sur le même principe que la division euclidienne sur les entiers, nous pouvons effectuer la division euclidienne sur les polynômes.
C'est-à-dire, tant que le polynôme à diviser est de plus haut degré que le diviseur, on continue à éliminer le coefficient de plus haut degré du polynôme à diviser.

Exemple de $x^4 + x^3 + x$ divisé par $x^2 + 1$ dans $\mathbb{F}_2[X]$:

$$\begin{array}{rrrr|l}
x^4 & +x^3 & & +x & x^2+1 \\
\cline{5-5}
x^4 & & +x^2 & & x^2+x+1 \\
\cline{1-3}
 & x^3 & +x^2 & +x & \\
\cline{2-4}
 & x^3 & & +x & \\
\cline{2-4}
 & & x^2 & & \\
\cline{3-4}
 & & x^2 & +1 & \\
\cline{3-4}
 & & & 1 &
\end{array}$$

On peut alors écrire : $x^4 + x^3 + x = (x^2 + x + 1)(x^2 + 1) + 1$
Où le quotient $q = x^2 + x + 1$ et le reste $r = 1$
Soient $h, g, q, r \in \mathbb{F}_2[X]$ tels que $h = qg + r$
Si $r = 0$, alors on dit que le polynôme g divise le polynôme h.

2.3.4 PGCD

En se basant sur la division euclidienne on peut aussi calculer le PGCD de deux polynômes de $\mathbb{F}_2[X]$.
C'est-à-dire qu'à chaque itération on divise le polynôme de plus haut degré par celui de plus petit degré jusqu'à obtenir 0.
On peut calculer le PGCD de deux polynômes de $\mathbb{F}_2[X]$ de la même manière que le PGCD classique entre les entiers.
La différence est que pour comparer deux polynômes, on prend comme plus grand polynôme celui de plus haut degré.
Exemple :

Calculons le PGCD de $x^4 + x^3 + x^2 + 1$ et de $x^3 + x^2 + x + 1$:

$$\begin{array}{rcrcrcr} x^4 + x^3 + x^2 + 1 & = & x & \times & (x^3 + x^2 + x + 1) & + & x + 1 \\ x^3 + x^2 + x + 1 & = & (x^2 + 1) & \times & (x + 1) & + & 0 \end{array}$$

On a ici : $PGCD(x^4 + x^3 + x^2 + 1, x^3 + x^2 + x + 1) = x + 1$

2.3.5 Identité de Bezout

L'identité de Bezout est vérifiée de la même manière pour les polynômes de $\mathbb{F}_2[X]$ que pour les entiers.
$\forall g, h \in \mathbb{F}_2[X] \quad \exists u, v \in \mathbb{F}_2[X] \quad PGCD(g, h) = ug + vh$
Pour obtenir u et v, on va calculer les u_i, v_i pour chaque itération.
On définit au début :

$$\begin{array}{ll} u_0 = 1 & u_1 = 0 \\ v_0 = 0 & v_1 = 1 \end{array}$$

Maintenant on va calculer le PGCD de $a, b \in \mathbb{F}_2[X]$.
On pose $r_0 = a$ et $r_1 = b$.
Tant que $r_k \neq 0$
q_k est le quotient de la division euclidienne de r_{k-1} et r_{k-2}
$r_k = r_{k-2} - q_k \times r_{k-1}$
$u_k = u_{k-2} - q_k \times u_{k-1}$
$v_k = v_{k-2} - q_k \times v_{k-1}$
Lorsque r_k est le premier polynôme à valoir 0, on a alors $u = u_{k-1}$ et $v = v_{k-1}$.

Chapitre 3

Polynômes irréductibles de $\mathbb{F}_2[X]$

3.1 Définition

Un polynôme est dit irréductible dans $\mathbb{F}_2[X]$ s'il n'est divisible que par 1 et par lui-même.
Exemple :
x^2+1 n'est pas irréductible, en effet : $x^2+1=(x+1)(x+1)$, il est donc divisible par $x+1$.
x^2+x+1 est irréductible, en effet seuls 1 et x^2+x+1 divisent x^2+x+1.

3.2 Recherche de polynômes irréductibles

3.2.1 Crible d'Eratosthène

Une première approche pour trouver tous les polynômes irréductibles a été l'utilisation du Crible d'Eratosthène.
Ma méthode est de sélectionner un degré d dont on connaît tous les irréductibles de degré inférieur.
On fait la liste des polynômes possibles pour le degré d.
On multiplie tous les polynômes irréductibles de degrés au minimum 1 strictement inférieur à d tels que la somme de leurs degrés fasse d.
On commence avec x et $x+1$ irréductibles.
Par exemple :
Pour le degré 2 :

x^2 $= x \times x$
x^2+x $= x \times (x+1)$
x^2+1 $= (x+1) \times (x+1)$
x^2+x+1 est irréductible

Pour le degré 3 :

$x^3 = x \times x \times x$
$x^3 + x^2 = x \times x \times (x+1)$
$x^3 + x = x \times (x+1) \times (x+1)$
$x^3 + x^2 + x = x \times (x^2 + x + 1)$
$x^3 + x^2 + x + 1 = (x+1) \times (x+1) \times (x+1)$
$x^3 + 1 = (x+1) \times (x^2 + x + 1)$
$x^3 + x + 1$ est irréductible
$x^3 + x^2 + 1$ est irréductible

Par cette recherche des polynômes irréductibles dans $\mathbb{F}_2[X]$, on obtient le nombre de polynômes par degré suivant :

Degré des polynômes	Nombre de polynômes irréductibles
0	1
1	2
2	1
3	2
4	3
5	6
6	9
7	18
8	30
9	56
10	99
11	186
12	335
13	630
14	1161
15	2182
16	4080
17	7710
18	14532
19	27594
20	52377

On peut obtenir plus d'informations sur les polynômes irréductibles de $\mathbb{F}_2[X]$ en lançant une recherche avec les premières valeurs de cette séquence à l'aide de 'The On-Line Encyclopedia of Integer Sequences® (OEIS®)'

3.2.2 Test de probable irréductibilité

On a la propriété que pour tout $q \in \mathbb{N}^*$, $x^{2^q} - x$ est le produit de tous les polynômes irréductibles de $\mathbb{F}_2[X]$ dont le degré d divise q.
On peut utiliser cette propriété pour effectuer une première vérification d'irréductibilité. Soit $p \in \mathbb{F}_2[X]$ et $d = deg(p)$, si d est premier et ni x, ni $x+1$ ne divise p et que p divise $x^{2^q} - x$, alors p est irréductible dans $\mathbb{F}_2[X]$.
Soit $p \in \mathbb{F}_2[X]$, cherchons si p est irréductible dans $\mathbb{F}_2[X]$.

première approche, on peut calculer les x^{2^i} modulo p pour $i \in [\![1, deg(p)]\!]$
Si p est divisible par x ou par $x+1$ il n'est pas irréductible.
Si on rencontre un x^{2^i} modulo p qui vaut 0 ou 1, p n'est pas irréductible.
Si $x^{2^{deg(p)}} \equiv x\,[p]$, on a alors $x^{2^q} - x \equiv 0\,[p]$, donc p est probablement irréductible ou irréductible si $deg(p)$ est premier.
On calcule les x^{2^i} modulo p ce qui permet de rester dans une plage mémoire raisonnable en complexité mémoire $O(deg(p))$ d'un point de vue machine.

Exemple pour $x^4 + x + 1$:

x^{2^0}	$\equiv x$	$[x^4 + x + 1]$
x^{2^1}	$\equiv x \times x \equiv x^2$	$[x^4 + x + 1]$
x^{2^2}	$\equiv x^2 \times x^2 \equiv x^4 \equiv x + 1$	$[x^4 + x + 1]$
x^{2^3}	$\equiv (x+1)(x+1) \equiv x^2 + 1$	$[x^4 + x + 1]$
x^{2^4}	$\equiv (x^2+1)(x^2+1) \equiv x^4 + 1 \equiv x$	$[x^4 + x + 1]$

$x^{2^4} \equiv x\,[x^4 + x + 1]$, $x^4 + x + 1$ est probablement irréductible.

3.2.3 Test d'irréductibilité

Soit $p \in \mathbb{F}_2[X]$.
On teste si p est probablement irréductible avec le procédé énoncé dans la partie 'Test de probable irréductibilité'.
Si $deg(p)$ est premier, p est irréductible.
Sinon on lance l'Algorithme de Berlekamp (en section suivante).
Si on a réussi une factorisation et que p est seul facteur, alors p est irréductible.

Pour vérifier le fonctionnement de ce test de primalité, lors de l'utilisation du Crible d'Eratosthène, on vérifie le résultat avec ce test de primalité, qui fonctionne à 100%¸ jusqu'aux polynômes de degré au plus 20.

3.2.4 Recherche par génération aléatoire

En première approche, on générait un polynôme p de $\mathbb{F}_2[X]$ d'un degré choisi dont les valeurs des coefficients étaient définies aléatoirement.
Puis on testait son irréductibilité avec le test d'irréductibilité.
Le problème de ce test est que beaucoup de polynômes n'était pas nécessairement irréductibles et qu'il était possible d'en extraire des polynômes irréductibles de degré plus petit.
Ce qui fait, que j'ai choisi d'appliquer l'Algorithme de Berlekamp récursivement en cas de succès de factorisation pour trouver les polynômes dont p était le produit et appliquer le procédé jusqu'à trouver un diviseur irréductible.
Avec ce procédé, on a pu trouver des polynômes irréductibles dont le degré dépasse 10000 (maximum trouvé : degré 11457).

3.3 Algorithme de Berlekamp

3.3.1 Dérivée formelle

Pour l'utiliser dans l'Algorithme de Berlekamp nous allons introduire la dérivée formelle d'un polynôme de $\mathbb{F}_2[X]$.
Soit $p \in \mathbb{F}_2[X]$ tel qu'il existe $p+1 \in \mathbb{N}$ scalaires $\alpha_0, \alpha_1, \ldots, \alpha_p \in \mathbb{F}_2$
tels que pour tout x, $p(x) = \sum_{i=0}^{p} \alpha_i x^i$.
On définit la dérivée formelle de p, p' telle que :
$p'(x) = \sum_{i=1}^{p} i\alpha_i x^{i-1}$.
Ici on considère que le scalaire i est la répétition de l'opération d'addition qui est le 'xor', ce qui annule toutes les coefficients d'indice $i \equiv 0[2]$, d'où :
$p'(x) = \sum_{i=0}^{\lfloor \frac{p-1}{2} \rfloor} \alpha_{2i+1} x^{2i}$.

3.3.2 Matrice de Berlekamp

On va dans un premier temps construire la matrice de Berlekamp.
La matrice sera de taille $deg(p) \times deg(p)$.
La colonne $i \in [\![1, deg(p)]\!]$ de la matrice est composée des coefficients de $x^{2(i-1)} - x^{i-1}$ modulo p.
Exemple pour $x^4 + x$:

$$\begin{array}{lll} x^0 - x^0 & \equiv 0 & [x^4+x] \\ x^2 - x & \equiv x^2 + x & [x^4+x] \\ x^4 - x^2 & \equiv x^2 + x & [x^4+x] \\ x^6 - x^3 & \equiv 0 & [x^4+x] \end{array}$$

Ce qui nous fait la matrice suivante :

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Pour avoir un second exemple, pour $x^4 + x^2 + x + 1$, la matrice est la suivante :

$$\begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

3.3.3 Noyau de la matrice

Nous allons maintenant appliquer le pivot de Gauss sur les colonnes de la matrice.
Le nombre de colonnes nulles est le nombre de facteurs irréductibles que l'on peut extraire de p.

Pour extraire les bases du noyau de la matrice, j'ai choisi de créer une nouvelle matrice identité sur laquelle on applique les mêmes opérations avec le pivot de Gauss que sur la matrice de Berlekamp.
Exemple pour la matrice de $x^4 + x$:

$$\left(\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ \hline 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right) \Longrightarrow_{\substack{C_1 \longleftarrow C_2 \\ C_2 \longleftarrow C_3 \\ C_3 \longleftarrow C_1}} \left(\begin{array}{cccc} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ \hline 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right) \Longrightarrow_{C_2 \longleftarrow C_1 + C_2} \left(\begin{array}{cccc} 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ \hline 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right)$$

On obtient ici les bases suivantes : $\{x^2 + x, 1, x^3\}$

3.3.4 Principe

Pour l'Algorithme de Berlekamp on veut factoriser un polynôme p.
On calcule sa dérivée formelle p'.
Si $p' = 0$, alors p est à racines multiples, on s'arrête.
Si $PGCD(p, p') \neq 1$, alors on renvoie $\left\{PGCD(p, p'), \frac{p}{PGCD(p,p')}\right\}$ et on s'arrête (On voit la division ici comme la division euclidienne de deux polynômes).
On calcule la matrice de Berlekamp pour p.
On extrait les bases de la matrice.
On note R l'ensemble des facteurs trouvés de p.
Pour toute base b_i de la matrice.
On calcule les $t_{i,\alpha} = PGCD(p, b_i - \alpha)$ pour tout $\alpha \in \mathbb{F}_2$, autrement dit $\alpha \in \{0, 1\}$.
Pour tout $r_j \in R$, si $PGCD(t_{i,\alpha}, r_j) \neq 1$:
On divise celui d'entre $t_{i,\alpha}$ et r_j par $PGCD(t_{i,\alpha}, r_j)$ et on le remplace uniquement si le résultat de la division est différente de 1.
Une fois tous les éléments de R traités, on ajoute $t_{i,\alpha}$ à R si $t_{i,\alpha} \notin R$.

Exemple pour $x^4 + x$:
la dérivée formelle de $x^4 + x$ est $1 \neq 0$.
$PGCD(x^4 + x, 1) = 1$, on lance alors la factorisation par le calcul de la matrice.
Les calculs de la Matrice et des bases du noyau sont réalisés en exemples plus haut.
On a alors $B = \{x^2 + x, 1, x^3\}$ l'ensemble des bases.

$$\begin{array}{lll} t_{1,0} & = PGCD(x^4 + x, x^2 + x) & = x^2 + x \\ t_{1,1} & = PGCD(x^4 + x, x^2 + x + 1) & = x^2 + x + 1 \\ t_{2,0} & = PGCD(x^4 + x, 1) & = 1 \\ t_{2,1} & = PGCD(x^4 + x, 0) & = x^4 + x \\ t_{3,0} & = PGCD(x^4 + x, x^3) & = x \\ t_{3,1} & = PGCD(x^4 + x, x^3 + 1) & = x^3 + 1 \end{array}$$

On commence à $R = \emptyset$
On ajoute $t_{1,0} = x^2 + x$ à R car $x^2 + x \notin R$
D'où $R = \{x^2 + x\}$

Pour $t_{1,1} = x^2 + x + 1$, on regarde son PGCD avec chaque élément de R :
$PGCD(x^2 + x, x^2 + x + 1) = 1$ donc rien à changer dans R.
Tous les éléments de R sont traités, $x^2 + x + 1 \notin R$, alors on l'ajoute à R.
D'où $R = \{x^2 + x, x^2 + x + 1\}$

$t_{2,0} = 1$, on passe au suivant.

$PGCD(x^2 + x, t_{2,1}) = PGCD(x^2 + x, x^4 + x) = x^2 + x$
$t_{2,1}$ est de plus grand degré, $\frac{t_{2,1}}{x^2+x} = \frac{x^4+x}{x^2+x} = x^2 + x + 1$
$x^2 + x + 1$ appartient déjà à R.
$PGCD(x^2 + x + 1, x^2 + x + 1) = x^2 + x + 1$
La division euclidienne d'un des $x^2 + x + 1$ par $x^2 + x + 1$ donnera 1, rien à modifier

$PGCD(x^2 + x, t_{3,0}) = PGCD(x^2 + x, x) = x$
$x^2 + x$ est de plus grand degré, $\frac{x^2+x}{x} = x + 1$
On remplace $x^2 + x$ par $x + 1$ dans R, d'où :
$R = \{x + 1, x^2 + x + 1\}$
$PGCD(x^2 + x + 1, t_{3,0}) = PGCD(x^2 + x + 1, x) = 1$.
$x \notin R$, alors on ajoute x à R :
$R = \{x + 1, x^2 + x + 1, x\}$

$PGCD(x + 1, t_{3,1}) = PGCD(x + 1, x^3 + 1) = x + 1$
$t_{3,1}$ est de plus grand degré, $\frac{t_{3,1}}{x+1} = \frac{x^3+1}{x+1} = x^2 + x + 1$
$x^2 + x + 1$ appartient déjà à R.
$PGCD(x^2 + x + 1, x^2 + x + 1) = x^2 + x + 1$, rien à modifier
$PGCD(x, x^2 + x + 1) = 1$, rien à modifier

On a alors $R = \{x + 1, x^2 + x + 1, x\}$.
On a donc $x^4 + x = x(x + 1)(x^2 + x + 1)$.

Exemple avec $x^5 + x^4 + x^2 + x$:
Sa dérivée formelle est $x^4 + 1 \neq 0$
$PGCD(x^5 + x^4 + x^2 + x, x^4 + 1) = x^2 + 1 \neq 1$
On a $\frac{x^5+x^4+x^2+x}{x^2+1} = x^3 + x^2 + x$
On renvoie alors $\{x^2 + 1, x^3 + x^2 + x\}$
On remarque ici que ce ne sont pas des facteurs irréductibles de $x^5 + x^4 + x^2 + x$ même si c'était le cas dans l'exemple précédent.

3.4 Utilisations

3.4.1 Calcul d'inverse

On peut utiliser les polynômes irréductibles pour calculer ce que nous nommerons un 'inverse' par la multiplication d'un polynôme.
Soit $g, p \in \mathbb{F}_2[X]$ tels que $PGCD(p, g) = 1$.
Nous noterons $p^{-1} \in \mathbb{F}_2[X]$ l'inverse de p par g le polynôme qui vérifie :

$$p \times p^{-1} \equiv 1\,[g]$$

Pour calculer l'inverse d'un polynôme de manière générale nous utiliserons l'identité de Bezout.
C'est-à-dire que nous vérifierons que $PGCD(p, g) = 1$ et nous obtenons $u, v \in \mathbb{F}_2[X]$ tels que $u \times p + v \times g = 1$.
Ce qui nous fait : $u \times p = 1 - v \times g$ et donc $u \times p \equiv 1\,[g]$.
u est alors l'inverse de p par g.

Exemple : trouver l'inverse de $x + 1$ par $x^2 + x + 1$ s'il existe.
$PGCD(x + 1, x^2 + x + 1) = 1 = x(x + 1) + x^2 + x + 1$.
On a $u = x$, x est alors l'inverse de $x+1$ par x^2+x+1, en effet : $x(x+1) \equiv 1\,[x^2 + x + 1]$.

3.4.2 Construction des corps $\mathbb{F}_{2^q}$

On peut construire des corps finis plus grands à partir de $\mathbb{F}_2[X]$.
C'est-à-dire qu'on choisit un polynôme irréductible p dans $\mathbb{F}_2[X]$.
On va pouvoir construire $\mathbb{F}_{2^q} = \mathbb{F}_2[X]/p(X)$ tel que $q = deg(p)$.
Pour gérer informatiquement un polynôme de $\mathbb{F}_{2^q}[X]$ j'utilise une structure où les coefficients sont un tableau de polynômes de $\mathbb{F}_2[X]$ et je stocke aussi le polynôme irréductible qui engendre $\mathbb{F}_{2^q}$ pour effectuer les opérations sur les coefficients modulo ce polynôme.

L'addition et la soustraction de ce corps correspondent au 'xor', nous ferons nos calculs avec des polynômes de $\mathbb{F}_2[X]$ de la même manière mais modulo p.
La multiplication reste la même que dans $\mathbb{F}_2[X]$ mais modulo p.
De plus nous avons choisi p irréductible, on peut alors trouver un inverse pour tout polynôme de $\mathbb{F}_{2^q}$.
Par exemple : nous pouvons construire $\mathbb{F}_{16} = \mathbb{F}_2[X]/(X^4 + X + 1)$.

Pour faire le lien avec les entiers nous pourrons représenter les polynômes de $\mathbb{F}_2[X]$ par le mot composé de ses coefficients.
Par exemple : $x^4 + x + 1$ s'écrit 10011 avec cette convention.

3.4.3 Polynôme primitif

Pour déterminer si un polynôme p est primitif, on regarde s'il existe un élément du corps fini $\mathbb{F}_2[X]/p(X)$ dont les puissances pour la loi multiplicative engendre tous les éléments de ce corps fini, on dira que cet élément est primitif dans $\mathbb{F}_2[X]/p(X)$.

Soit α un élément primitif de $\mathbb{F}_2[X]/p(X)$, alors $\mathbb{F}_2[X]/p(X) = \left\{\alpha^i \in \mathbb{F}_2[X]/p(X) \quad i \in [\![1, 2^{deg(p)}]\!]\right\}$.
Dans les calculs des BCH par exemple, nous prendrons α tel que α est le représentant de x dans $\mathbb{F}_2[X]/p(X)$ et que α est un élément primitif de $\mathbb{F}_2[X]/p(X)$.

Déterminer si α est un élément primitif dans $\mathbb{F}_2[X]/p(X)$ et donc dire que p est primitif, on va utiliser le procédé suivant :
On calcule les α^i pour $i \in [\![1, 2^{deg(p)}]\!]$.
Si $\alpha^{2^{deg(p)}} \neq 1$, α n'est pas un élément primitif pour p.
Si il existe $j \in [\![1, 2^{deg(p)} - 1]\!]$ tel que $\alpha^j = 1$, α n'est pas un élément primtif pour p.

Exemple : on aimerait trouver un polynôme p primitif de $\mathbb{F}_2[X]$ tel que α représentant de x soit un élément primitif pour p.
$x^4 + x^3 + x^2 + x + 1$ est irréductible dans $\mathbb{F}_2[X]$, vérifions si α est un élément primitif pour $x^4 + x^3 + x^2 + x + 1$.
Nous prendrons la convention d'écrire les polynômes sous forme binaire pour nos calculs, d'où $x^4 + x^3 + x^2 + x + 1$ représenté par 11111.
Travaillons dans $\mathbb{F}_2[X]/(X^4 + X^3 + X^2 + X + 1)$:
$\alpha = 10$
$\alpha^2 = 100$
$\alpha^3 = 1000$
$\alpha^4 = 1111$
$\alpha^5 = 1$
α n'est pas un élément primitif pour $x^4 + x^3 + x^2 + x + 1$.

Intéressons nous à $x^4 + x + 1$, irréductible dans $\mathbb{F}_2[X]$ représenté par 10011.
Travaillons dans $\mathbb{F}_2[X]/(X^4 + X + 1)$:
$\alpha = 10$
$\alpha^2 = 100$
$\alpha^3 = 1000$
$\alpha^4 = 11$
$\alpha^5 = 110$
$\alpha^6 = 1100$
$\alpha^7 = 1011$
$\alpha^8 = 101$
$\alpha^9 = 1010$
$\alpha^{10} = 111$
$\alpha^{11} = 1110$
$\alpha^{12} = 1111$
$\alpha^{13} = 1101$
$\alpha^{14} = 1001$
$\alpha^{15} = 1$
On a alors que α est un élément primitif pour $x^4 + x + 1$.
Et de plus $x^4 + x + 1$ est primitif dans $\mathbb{F}_2[X]$

Chapitre 4

codes BCH

Les codes BCH de leurs auteurs R.C. Bose, D.K. Ray-Chaudluri et A. Hocquenghem sont des codes permettant de corriger des erreurs dans un message envoyé en supposant que le nombre d'erreur est inférieur à la capacité de correction prévue à la construction.

4.1 Construction

4.1.1 Choix du polynôme primitif de $\mathbb{F}_2[X]$

Dans le cas général, on se donne n la longueur d'un code BCH et $\mathbb{F}_q$ un corps fini tels que $PGCD(n, q) = 1$.
On détermine un entier m tel qu'il est le plus petit vérifiant la propriété suivante :

$$q^m \equiv 1\,[n]$$

Ceci dans le but de construire $\mathbb{F}_{q^m}$ tel qu'il contienne un élément primitif.
Nous travaillons à partir de $\mathbb{F}_2$, on pose alors $q = 2$.
Nous voulons α comme élément primitif de x dans $\mathbb{F}_{2^m}$.
On choisit alors un polynôme primitif p de degré m dont α est un élément primitif.
Exemple : nous voulons choisir un polynôme pour construire un code BCH de longueur 15.
Nous travaillons dans $\mathbb{F}_2$, 2 et 15 sont premiers entre eux, nous allons alors calculer les puissances successives de 2 modulo 15.
$2^1 \equiv 2\,[15]$
$2^2 \equiv 4\,[15]$
$2^3 \equiv 8\,[15]$
$2^4 \equiv 1\,[15]$
On trouve alors $m = 4$, on cherche alors un polynôme de degré 4 dont α est un élément primitif.
Dans nos exemple sur les polynômes primitifs, nous avions trouvé $x^4 + x + 1$ qui vérifie cette propriété.

4.1.2 Calcul du polynôme minimal

On se place dans le cas d'un code dit d-correcteur.
On définit le polynôme minimal $g \in \mathbb{F}_2[X]$ tel que pour tout $i \in [\![1, 2m]\!]$:

$$g(\alpha^i) = 0$$

Dans $\mathbb{F}_2$ on a la propriété remarquable que si $g(\alpha) = 0$, alors $g(\alpha^2) = g(\alpha)^2 = 0$.
Ce qui revient à trouver le polynôme g qui s'annule aux puissances impaires de α.
On va calculer les conjugués successifs des puissances impaires de α.
C'est-à-dire pour la puissance impaire i et $k > 0$ le plus petit entier tel que $2^k i \equiv i\,[n]$,
On calcule l'ensemble $C_i = \left\{\alpha^{2^j i} \quad j \in [\![0, k-1]\!]\right\}$ que l'on nomme aussi classe cyclotomique de α^i.
On note M_i le polynôme minimal de α^i, autrement dit $M_i(x) = \prod_{\delta \in C_i} (x - \delta)$
On calcule g tel que $g(x) = PPCM(M_1(x), M_3(x), \ldots, M_{2d-3}(x), M_{2d-1}(x))$
Pour déterminer les polynômes minimaux M_i :
Si $i = 1$, c'est le polynôme p.
Sinon, on se propose deux méthodes :

- On construit le corps fini $\mathbb{F}_2[X]/p(X)$ et on développe M_i pour obtenir ses coefficients.
- On pose un système pour trouver M_i dont le degré est le cardinal de C_i et on applique par exemple la méthode de Cramer pour résoudre le système.

J'ai implémenté les deux méthodes dans le projet, initialement la seconde, puis la première plus tard.

Un fois le polynôme g trouvé on calculera un polynôme h dit polynôme de contrôle tel que :

$$g(x)h(x) = x^n - 1$$

On notera le BCH construit BCH(n, $n - deg(g)$, d)

Exemple de construction d'un code BCH de longueur 15 2-correcteur :
On choisit le polynôme $x^4 + x + 1$ à partir duquel on va construire $\mathbb{F}_{16} = \mathbb{F}_2[X]/(X^4 + X + 1)$.
Puisque notre code est 2-correcteur, on va calculer les puissances impaires de α jusqu'à $2d - 1 = 2 \times 2 - 1 = 3$.
C'est-à-dire que nous allons chercher g tel que :

$$g(\alpha) = g(\alpha^3) = 0$$

$C_1 = \{\alpha, \alpha^2, \alpha^4, \alpha^8\}$
$C_3 = \{\alpha^3, \alpha^6, \alpha^{12}, \alpha^9\}$
$M_1(x) = (x - \alpha)(x - \alpha^2)(x - \alpha^4)(x - \alpha^8) = x^4 + x + 1$ par définition.
$M_3(x) = (x - \alpha^3)(x - \alpha^6)(x - \alpha^9)(x - \alpha^{12}) = a + bx + cx^2 + dx^3 + x^4$.
Par définition α^3 est racine de M_3, d'où (nous prenons la notation binaire pour les coefficients calculés) :

$[0001]a + [1000]b + [1100]c + [1010]d + [1111] = [0000]$.
Ce qui équivaut à :
$[0001]a + [1000]b + [1100]c + [1010]d = [1111]$.
Par la méthode de Cramer nous construisons la matrice suivante :

$$M = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

En calculant le déterminant de M, on a $Det(M) = 1$
On obtient les coefficients en remplaçant la colonne qui leur est associée par la valeur du vecteur à obtenir, ce qui nous donne :

$$a = \frac{Det\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}}{Det(M)} = 1$$

$$b = \frac{Det\begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{pmatrix}}{Det(M)} = 1$$

$$c = \frac{Det\begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}}{Det(M)} = 1$$

$$d = \frac{Det\begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix}}{Det(M)} = 1$$

D'où $M_3(x) = x^4 + x^3 + x^2 + x + 1$.
On construit alors g tel que $g(x) = PPCM(M_1(x), M_3(x)) = (x^4 + x + 1)(x^4 + x^3 + x^2 + x + 1)$
On calcule h tel que $h(x) = \frac{x^{15}-1}{g(x)} = x^7 + x^6 + x^4 + 1$.
On note le BCH construit BCH(15, 7, 2).

4.2 Codage

Nous noterons t_m la taille du message à coder par le code BCH.
On dispose d'un code BCH(n, t_m, d).
Pour coder un message m représenté sous forme d'un mot binaire de taille inférieure ou égale à t_m :
On écrit ce message sous forme de polynôme à coefficients dans $\mathbb{F}_2$.
On calcule un polynôme r comme étant le reste de la division euclidienne de $m(x) \times x^{deg(g)}$ par g.
D'où u le message encodé tel que $u(x) = m(x) \times x^{deg(g)} + r(x)$.

Exemple : codage de 0111001 avec BCH(15, 7, 2)
$m(x) = x^5 + x^4 + x^3 + 1$
$r(x) = m(x) \times x^8 \equiv x^{13} + x^{12} + x^{11} + x^8 \equiv x^7 + x$
$u(x) = m(x) \times x^8 + r(x) = x^{13} + x^{12} + x^{11} + x^8 + x^7 + x$
$u = 11100110000010$

4.3 Décodage

On reçoit un message v tel qu'il existe un polynôme d'erreurs e tel que $v(x) = u(x) + e(x)$.

4.3.1 Vérification

Pour vérifier l'existence d'erreurs, on calcule $v(x)h(x)$ modulo $x^n - 1$ où h est notre polynôme de contrôle.
Si $v(x)h(x) \equiv 0\,[x^n - 1]$, il n'y a pas d'erreurs.
Sinon on lance le processus de correction des erreurs.

4.3.2 Correction des erreurs

Calcul des syndromes

Pour le calcul des syndromes, on calcule les syndromes s_i pour $i \in [\![1, 2d]\!]$ de la manière suivante :

$$s_i = v(\alpha^i)$$

Calcul du polynôme localisateur

On note le polynôme localisateur d'erreurs σ tel que :

$$\sigma(x) = 1 + \sum_{i=1}^{t} \sigma_i x^i$$

Pour trouver sigma on se propose deux méthodes :
— La méthode due à Peterson qui revient à réaliser un pivot de gauss

— Résoudre un problème d'interpolation

Méthode due à Peterson

Par convention on pose $s_0 = 1$.
On construit une matrice de taille $d \times (d+1)$ des syndromes pour effectuer un pivot de gauss :

$$\begin{pmatrix} s_0 & s_1 & \cdots & s_{d-1} & s_d \\ s_1 & s_2 & \cdots & s_d & s_{d+1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ s_{d-1} & s_d & \dots & s_{2d-2} & s_{2d-1} \end{pmatrix}$$

Pour chaque colonne d'indice $i \in [\![1, d]\!]$, on va chercher à éliminer les coefficients de la colonne i en ne gardant 1 qu'à la ligne i.
De ce fait, on va obtenir une diagonale suivie d'un vecteur qui nous donnera les valeurs des coefficients du polynôme localisateur.
L'élimination de la colonne d'indice i se fait de la manière suivante :
Si le coefficient en (i, i) vaut 0, on recherche une ligne d'indice $k > i$ telle que le coefficient en (k, i) soit différent de 0, s'il n'existe pas de k permettant cette opération, le système est sous-évalué, on relance avec $d - 1$.
On multiplie chaque ligne j par l'inverse dans $\mathbb{F}_2[X]/p(X)$ du coefficient d'indice (j, i) s'il existe sinon on passe à la suivante
Pour toute ligne j telle que $i \neq j$, $L_j \longleftarrow L_j + L_i$ ce qui élimine les coefficients de la colonne i sauf à la ligne i.
On passe à la colonne suivante.
Exemple sur la première colonne :
On fait la multiplication par l'inverse :

$$\begin{pmatrix} 1 & s_1 s_0^{-1} & \cdots & s_{d-1} s_0^{-1} & s_d s_0^{-1} \\ 1 & s_2 s_1^{-1} & \cdots & s_d s_1^{-1} & s_{d+1} s_1^{-1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & s_d s_{d-1}^{-1} & \dots & s_{2d-2} s_{d-1}^{-1} & s_{2d-1} s_{d-1}^{-1} \end{pmatrix}$$

On élimine les coefficients de la première colonne :

$$\begin{pmatrix} 1 & s_1 s_0^{-1} & \cdots & s_{d-1} s_0^{-1} & s_d s_0^{-1} \\ 0 & s_2 s_1^{-1} + s_1 s_0^{-1} & \cdots & s_d s_1^{-1} + s_{d-1} s_0^{-1} & s_{d+1} s_1^{-1} + s_d s_0^{-1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & s_d s_{d-1}^{-1} + s_1 s_0^{-1} & \dots & s_{2d-2} s_{d-1}^{-1} + s_{d-1} s_0^{-1} & s_{2d-1} s_{d-1}^{-1} + s_d s_0^{-1} \end{pmatrix}$$

Au final après toutes les éliminations, on obtient la matrice suivante :

$$\begin{pmatrix} 1 & 0 & \cdots & 0 & \sigma_1 \\ 0 & 1 & \cdots & 0 & \sigma_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & \sigma_d \end{pmatrix}$$

J'ai initialement essayé de généraliser une méthode due à Peterson, puis implémenté ce pivot de gauss pour finalement préférer la méthode suivante.

Interpolation

On peut aussi résoudre le problème autrement.
Pour cela on construit le polynôme des syndromes s tel que :

$$s(x) = \sum_{i=0}^{2d-1} s_{i+1} x^i$$

On peut trouver σ le polynôme localisateur d'erreurs tel qu'il existe w un polynôme de degré inférieur à d tel que :

$$\sigma(x)s(x) + v(x)x^{2d} = w(x)$$

Le Théorème de Bezout nous donne qu'il existe $k \in \mathbb{N}$ tel qu'on peut trouver u_k, v_k et r_k les polynômes tels que :

$$u_k(x)x^{2d} + v_k(x)s(x) = r_k(x)$$

On a alors $\sigma = v_k$ et $w = r_k$.
Pour trouver σ j'ai alors implémenté une variante du calcul du PGCD pour lequel la condition est qu'on obtient u_k et v_k lorsque que r_k est de degré inférieur à notre taux de correction d.
Ceci est la méthode que j'utilise dans mon programme pour trouver le polynôme localisateur d'erreurs.

Correction des erreurs par Chien search

Nous avons la propriété que pour notre polynôme de localisation d'erreurs, s'il y a une erreur à corriger en position i :

$$\sigma(\alpha^i) = 0$$

Pour un BCH de longueur n nous itérons pour $i \in [\![0, d-1]\!]$.
Si $\sigma(\alpha^i) = 0$ alors on corrige l'erreur, c'est-à-dire que dans notre cas on corrige le bit en question.

4.3.3 Récupération du message

Pour récupérer le message, une fois corrigé, nous calculons m le polynôme tel que $m(x)$ est le quotient de la division euclidienne de $u(x)$ par x^{n-t_m}

Chapitre 5

QR codes

Les QR codes (Quick Response Codes) est un type de codes barres à deux dimensions (aussi nommé code matriciel), initialement créés par Denso-Wave en 1994, on les trouve un peut partout aujourd'hui sous la norme ISO/IEC 18004 :2006(E).

5.1 Construction

5.1.1 Versions

Il existe 40 versions pour un QR code standard.
Dans le but de programmer la construction de QR codes, j'ai implémenté la construction et le décodage des deux premières versions.
On peut obtenir la taille de l'image à fabriquer par la formule suivante :

$$taille\_image = 17 + 4 \times version$$

On considère que les bits de données sont représentés sur un pixel, il sera possible d'agrandir l'image après fabrication.
Avant toute écriture de données on dessine les patterns d'alignement.
On place tout d'abord les patterns principaux aux coins de l'image sauf celui en bas à droite qu'on relie par des lignes en pointillés de la manière suivante :

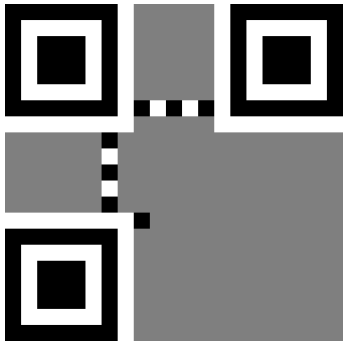


FIGURE 5.1 – *Base d'un QR code version 1*

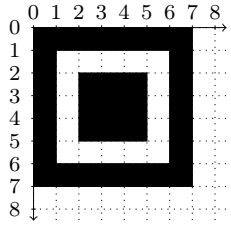


FIGURE 5.2 – *Pattern à avec échelle*

Pour les QR codes de version supérieure à 2, on ajoute des patterns d'alignement supplémentaires de forme :

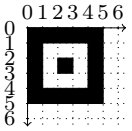


FIGURE 5.3 – *Pattern d'alignement à avec échelle*

Pour les placer, si la version est entre 2 et 6, on place un pattern d'alignement à la position $(4 \times (version + 2), 4 \times (version + 2))$.
Si la version est entre 7 et 13, on prend $(i, j) \in [\![0, 2]\!]^2$,
sauf si $(i = 0 \quad et \quad (j = 0 \quad ou \quad j = 2)) \quad ou \quad (i = 2 \quad et \quad j = 0)$,
on place un pattern d'alignement aux coordonnées :
$(4 + (i \times 2) \times (version + 1), 4 + (j \times 2) \times (version + 1))$
Ce qui nous donne les exemples suivants :

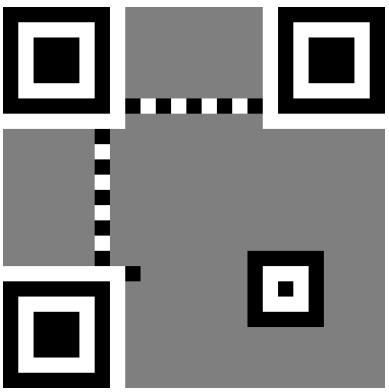


FIGURE 5.4 – *Base d'un QR code version 2*

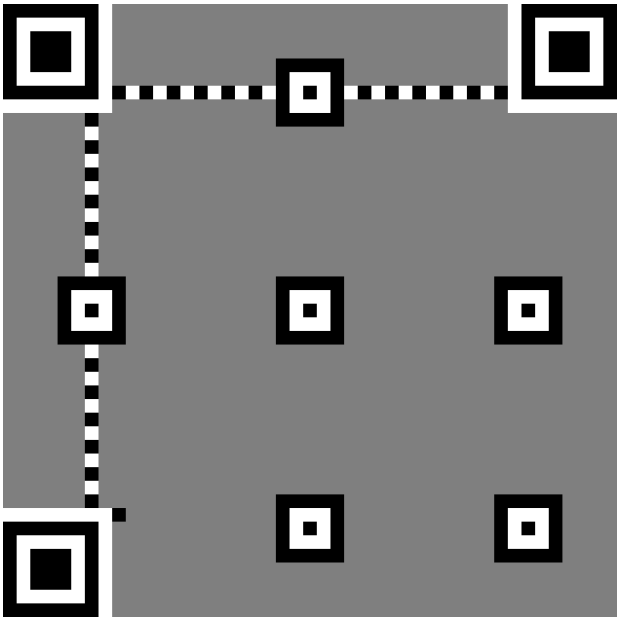


FIGURE 5.5 – *Base d'un QR code version 7*

J'ai utilisé une formule pour simplifier le placement des patterns d'alignement mais on peut retrouver leur position dans le tableau suivant :

Version	Position des patterns d'alignement						
1							
2	4	16					
3	4	20					
4	4	24					
5	4	28					
6	4	32					
7	4	20	36				
8	4	22	40				
9	4	24	44				
10	4	26	48				
11	4	28	52				
12	4	30	56				
13	4	32	60				
14	4	24	44	64			
15	4	24	46	68			
16	4	24	48	72			
17	4	28	52	76			
18	4	28	54	80			
19	4	28	56	84			
20	4	32	60	88			
21	4	26	48	70	92		
22	4	24	48	72	96		
23	4	28	52	76	100		
24	4	26	52	78	104		
25	4	30	56	82	108		
26	4	28	56	84	112		
27	4	32	60	88	116		
28	4	24	48	72	96	120	
29	4	28	52	76	100	124	
30	4	24	50	76	102	128	
31	4	28	54	80	106	132	
32	4	32	58	84	110	136	
33	4	28	56	84	112	140	
34	4	32	60	88	116	144	
35	4	28	52	76	100	124	148
36	4	22	48	74	100	126	152
37	4	26	52	78	104	130	156
38	4	30	56	82	108	134	160
39	4	24	52	80	108	136	164
40	4	28	56	84	112	140	168

5.1.2 Format

Dans le format d'un QR code on trouve deux informations :
— Le taux de correction

— Le masque du QR code

Le taux de correction est symbolisé par une lettre, à laquelle on associe un taux de correction.

Dans le format on représente le taux de correction choisi par l'écriture de 2 bits.

Lettre	Taux de correction (en %)	Représentation binaire
L	7	01
M	15	00
Q	25	11
H	30	10

Le masque est représenté par un entier sur 3 bits.

Le masque sera appliqué sur les données après codage comme un 'xor' sur les couleurs.

Pour les différentes valeurs du masque on donne les formules suivantes où i est l'indice de la colonne et j l'indice de la ligne :

Numéro du masque	formule du masque
0	(i + j) % 2 == 0
1	i % 2 == 0
2	j % 3 == 0
3	(i + j) % 3 == 0
4	((i / 2) + (j / 3)) % 2 == 0
5	(((i × j) % 2) + ((i × j) % 3)) == 0
6	(((i × j) % 2) + ((i × j) % 3)) % 2 == 0
7	(((i + j) % 2) + ((i × j) % 3)) % 2 == 0

Lorsque la formule associée au masque donne 1 on inverse la couleur aux coordonnées (i, j)

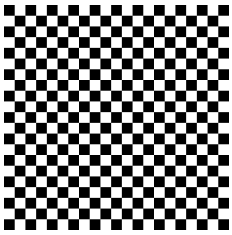


FIGURE 5.6 – *Masque 0*

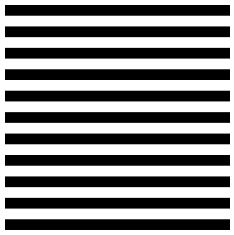


FIGURE 5.7 – *Masque 1*

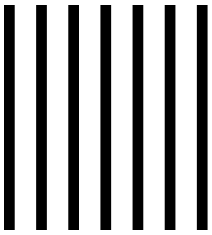


FIGURE 5.8 – *Masque 2*

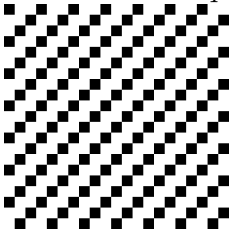


FIGURE 5.9 – *Masque 3*

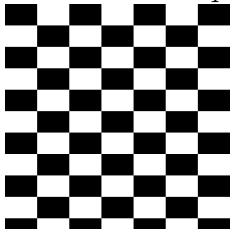


FIGURE 5.10 – *Masque 4*

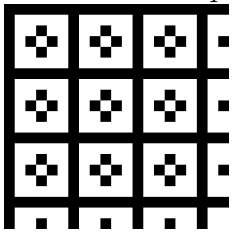


FIGURE 5.11 – *Masque 5*

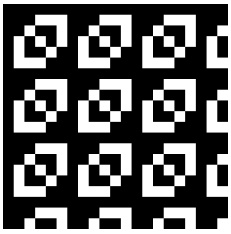


FIGURE 5.12 – *Masque 6*

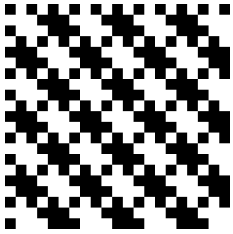


FIGURE 5.13 – *Masque 7*

Pour construire le format, on va se munir d'un BCH(15, 5, 3) dont $\mathbb{F}_{16}$ est engendré par $x^4 + x + 1$, noté en binaire 10011.
Notre message se composera de 5 bits, le format en 2 bits concaténé au numéro du masque en 3 bits.
Codé par le BCH, on obtient un mot de 15 bits que nous écrirons deux fois.
Avant d'écrire le mot obtenu on applique un 'xor' avec ce mot et le mot 101010000010010.
On écrit en noir les bits à 1 et en blanc les bits à 0.
On place le mot binaire final de la manière suivante :

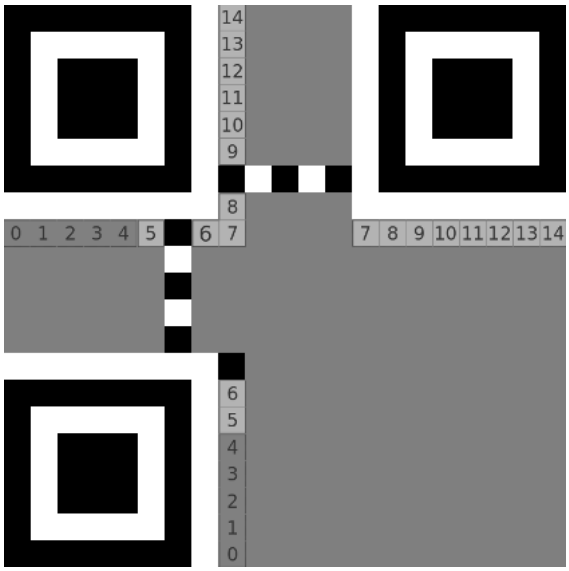


FIGURE 5.14 – *Emplacement du format*

En 0 les premiers bits du mot final.

Exemple : codage du format L1 :
Le mot associé à L est 01, le masque numéro 1 est représenté par 001.
Notre format est alors 01001.
Codé par notre BCH on obtient : 10011011100001.

Après application du ‘xor‘, le mot final est 10011011100001+101010000010010 = 111001011110011
Ce qui donne l'écriture du format de la manière suivante :

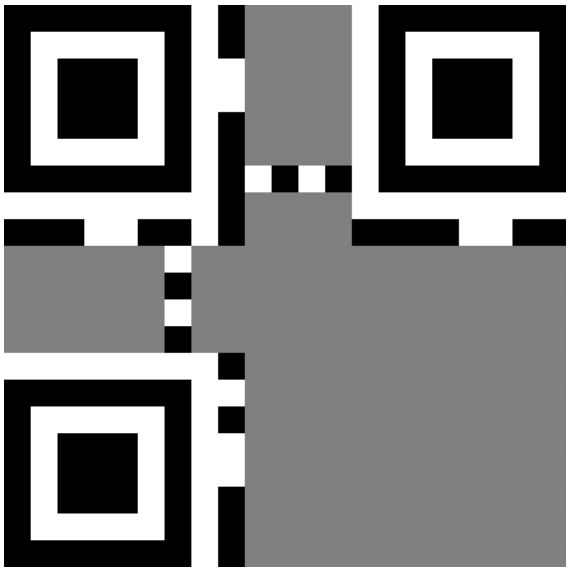


FIGURE 5.15 – *Exemple du format L1*

5.1.3 Format des données

Selon les données que l'on veut écrire dans le QR code, il existe plusieurs méthodes d'encodage pour réduire la taille des données si le message ne se compose que d'un nombre par exemple.
J'ai implémenté 3 formats différents :

- Le format Numérique : pour coder uniquement un nombre
- Le format Alphanumérique : pour coder majuscules et nombres ainsi que caractères spéciaux
- Le format Byte : pour coder les caractères de la table ASCII

On écrit le format sur 4 bits, avec les codes suivants :

Format	Mot associé en binaire
Numérique	0001
Alphanumérique	0010
Byte	0100

Suivi du nombre de caractères du message.
Selon la version du QR code le nombre de bits sur lequel on écrit le nombre de caractères du message varie, on le donne par le tableau suivant :

Version	Mode Numérique	Mode Alphanumérique	Mode Byte
1 à 9	10	9	8
10 à 26	12	11	16
27 à 40	14	13	16

Format Numérique

Pour le format numérique, on sépare notre message par groupes de 3 chiffres et le dernier la taille modulo 3 chiffres.
On considère chaque groupe comme un nombre en base 10 ayant 1 à 3 chiffres.
Si n est le nombre de chiffres du groupe, on l'écrit en binaire sur $4 + 3n$ bits.

Exemple de codage des données pour '1337042' :
On choisi le mode numérique, donc le message commencera par 0001
On a 7 caractères, on sélectionne un QR code version 1, donc écrit le nombre de caractères sur 10 bits, ce qui nous donne l'écriture de 7 sous la forme 00 0000 0111
On sépare '1337042' par groupes de 3 : 133 704 2.
$133_{(10)} = 10000101_{(2)}$ donc on écrit sur 10 bits : 00 1000 0101
$704_{(10)} = 1011000000_{(2)}$, ce qui donne sur 10 bits : 10 1100 0000
$2_{(10)} = 10_{(2)}$, ce qui donne sur 4 bits : 0010
On devra alors écrire dans le QR code le message suivant :
0001 0000000111 0010000101 1011000000 0010

Format Alphanumérique

Pour le format alphanumérique on sépare les caractères par groupes de 2.
On associe à chaque caractère une valeur dans la table suivante :

Car	Val	Car	Val	Car	Val	Car	Val	Car	Val	Car	Val	Car	Val	Car	Val
0	0	6	6	C	12	I	18	O	24	U	30	' '	36	.	42
1	1	7	7	D	13	J	19	P	25	V	31	$	37	/	43
2	2	8	8	E	14	K	20	Q	26	W	32	%	38	:	44
3	3	9	9	F	15	L	21	R	27	X	33	*	39		
4	4	A	10	G	16	M	22	S	28	Y	34	+	40		
5	5	B	11	H	17	N	23	T	29	Z	35	-	41		

On code le format et le nombre de caractères comme décrit dans la méthode générale.
On code chaque couple de caractères (u_1, u_2) où pour tout u caractère, $val(u)$ est la valeur dans le tableau de la manière suivante :
On calcule $val(u_1) \times 45 + val(u_2)$ puis on le convertit en binaire sur 11 bits.
Si on a un dernier caractère seul, on écrit sa valeur sur 6 bits.

Exemple de codage du mot 'MLV' :
Le mode est alphanumérique, d'où 0010 pour le format.
On code le message sur un QR code version 1, donc sur 9 bits, la taille est 3 caractères, d'où 0 0000 0011

On sépare le mot ‘MLV‘ en ML V :
$val(M) \times 45 + val(L) = 22 \times 45 + 21 = 1011_{(10)} = 11111111_{(2)}$, qui sera écrit sur 11 bits : 011 1111 11
$val(V) = 31_{(10)} = 11111_{(2)}$ qu'on codera sur 6 bits 01 1111
Ce qui nous fait le message final suivant :
0010 000000011 011111111 011111

Format Byte

Pour le format bit, cela revient à écrire les octets bruts tels qu'ils sont dans les données.
C'est-à-dire qu'on écrit l'indication sur 4 bits que c'est le mode byte qui est choisi, on écrit la taille des données puis on écrit les octets.

Exemple codage de ‘coucou‘ :
On écrit sur 4 bits que c'est le mode byte : 0100
Le nombre de caractères sur un QR code version 1, c'est-à-dire sur 8 bits : 0000 0110
Puis les données en binaire :
$c_{(ANSI)} = 1100011_{(2)}$
$o_{(ANSI)} = 1101111_{(2)}$
$u_{(ANSI)} = 1110101_{(2)}$
D'où le message final :
0100 00000110 01100011 01101111 01110101 01100011 01101111 01110101

5.1.4 Calculer le polynôme générateur

Les QR codes utilisent les codes Reed-Solomon dont le polynôme pour construire le corps fini $\mathbb{F}_{256}$ est représenté en binaire par 100011101
Autrement dit on travaille sur des polynômes à coefficients dans

$$\mathbb{F}_{256} = \mathbb{F}_2[X]/(X^8 + X^4 + X^3 + X^2 + 1)$$

On prend α comme élément primitif de $\mathbb{F}_{256}$ tel qu'il soit le représentant de x dans ce corps.
Pour un nombre de correction défini par n, on construit dans $\mathbb{F}_{256}[X]$ le polynôme générateur suivant :

$$g(X) = \prod_{i=0}^{n-1}(X - \alpha^i)$$

Exemple pour $n = 2$:
$g(x) = \prod_{i=0}^{2-1}(X - \alpha^i)$
$= (x - \alpha^0)(x - \alpha^1)$
Ici le addition est le ‘xor‘, donc la soustraction et l'addition sont la même opération.
$= (x + 1)(x + \alpha)$
$= x^2 + (\alpha + 1)x + \alpha$

Si on construit la table de $\mathbb{F}_2[X]/(X^8+X^4+X^3+X^2+1)$, on trouve $(\alpha+1)=\alpha^{25}$.
D'où on peut aussi écrire $g(x)=x^2+\alpha^{25}x+\alpha$

5.1.5 Construction d'un bloc

Pour construire les données à écrire on va avoir besoin des informations suivantes :

- La taille en octets des données du message
- La taille totale pour connaître le nombre d'octets ajoutés servant à la correction des erreurs

On va construire un bloc (t, t_m, c) défini par une taille totale t, un taille de données t_m et une capacité de correction de c erreurs de la manière suivante :
On reçoit des données de taille inférieure ou égale à t_m, on construit un polynôme de degré $t-1$ dont les coefficients de plus hauts degrés sont les valeurs des données (premières valeurs de la séquence pour les plus hauts coefficients).
On additionne à ce polynôme son reste modulo le polynôme générateur g.
Ce qui nous fait un bloc dont les $t_c = t - t_m$ coefficients sont les octets ajoutés pour la correction des erreurs.

Par exemple pour le bloc (10, 4, 2) :
t_c le nombre d'octets ajoutés pour la correction est $t_c = 10-4 = 6$
Ce qui veut dire que notre polynôme générateur ici est :
$g(x)=\prod_{i=0}^{6-1}(x-\alpha^i)$
$g(x)=(x-1)(x-\alpha)(x-\alpha^2)(x-\alpha^3)(x-\alpha^4)(x-\alpha^5)$
$g(x)=x^6+111111x^5+x^4+11011010x^3+100000x^2+11100011x+100110$
$g(x)=x^6+\alpha^{166}x^5+x^4+\alpha^{134}x^3+\alpha^5x^2+\alpha^{176}x+\alpha^{15}$
Si notre message est donné en hexadécimal de la manière suivante : 55 50 45 4D
On crée le polynôme suivant :
$1010101x^9+1010000x^8+1000101x^7+1001101x^6$
On a $1010101x^9+1010000x^8+1000101x^7+1001101x^6$ modulo $g(x)$
$=1000100x^5+1110100x^4+1001010x^3+10101100x^2+11011101x+110$
Ce qui nous donne au final le polynôme suivant en écrivant les coefficients en hexadécimal :
$55x^9+50x^8+45x^7+4Dx^6+44x^5+74x^4+4Ax^3+ACx^2+DDx+6$

5.1.6 Construction du message du QR code

Pour une version d'un QR code et un taux de correction d'erreurs, on peut coder le message par l'intermédiaire d'un ou plusieurs blocs.
Ci dessous le tableau donnant les différents blocs à construire :
On rappelle que pour un bloc (t, t_m, t_c), on construit un bloc de taille t dont les données occupent t_m octets/coefficients et t_c est le nombre d'erreurs corrigeables à titre d'information.
Le polynôme générateur d'un bloc sera de degré $t-t_m$.
Remarque : on a $t_c = \lfloor \frac{t-t_m}{2} \rfloor$

Description des blocs pour les 8 premières versions du QR code :

Version	Octets au total	Taux de correction	Nombre de blocs	Paramètres du bloc
1	26	L	1	(26, 19, 2)
		M	1	(26, 16, 4)
		Q	1	(26, 13, 6)
		H	1	(26, 9, 8)
2	44	L	1	(44, 34, 4)
		M	1	(44, 28, 8)
		Q	1	(44, 22, 11)
		H	1	(44, 16, 14)
3	70	L	1	(70, 55, 7)
		M	1	(70, 44, 13)
		Q	2	(35, 17, 9)
		H	2	(35, 13, 11)
4	100	L	1	(100, 80, 10)
		M	2	(50, 32, 9)
		Q	2	(50, 24, 13)
		H	4	(25, 9, 8)
5	134	L	1	(134, 108, 13)
		M	2	(67, 43, 12)
		Q	2	(33, 15, 9)
			2	(34, 16, 9)
		H	2	(33, 11, 11)
			2	(34, 12, 11)
6	172	L	2	(86, 68, 9)
		M	4	(43, 27, 8)
		Q	4	(43, 19, 12)
		H	4	(43, 15, 14)
7	196	L	2	(98, 78, 10)
		M	4	(49, 31, 9)
		Q	2	(32, 14, 9)
			4	(33, 15, 9)
		H	4	(39, 13, 13)
			1	(40, 14, 13)
8	242	L	2	(121, 97, 12)
		M	2	(60, 38, 11)
			2	(61, 39, 11)
		Q	4	(40, 18, 11)
			2	(41, 19, 11)
		H	4	(40, 14, 13)
			2	(41, 15, 13)

Pour créer le séquence finale :
Dans le QR code, la séquence a la forme ‘Données‘ suivies ‘Octets de correction d'erreurs‘.
Si elle se compose d'un seul bloc il suffira d'écrire le polynôme obtenu dans le QR code car on y trouve les données suivies des octets de correction des erreurs.
Si la séquence est divisée en plusieurs blocs, on va regrouper les données d'un côté et les octets de correction d'un autre.
Si on note k le nombre de blocs et B_i le bloc d'indice i.
Pour écrire le message on sépare chaque B_i en une séquence D_i les données et E_i la séquence associée au reste calculé par le polynôme générateur.
On note $D_{i,j}$ la valeur j-ème de D_i.
On écrit à la suite chaque première valeur de chaque D_i, puis une fois les premières valeurs traitées, on écrit chaque seconde valeur, ce qui nous donne le début de séquence suivante :
$D_{1,1}D_{2,1}\dots D_{k,1}D_{1,2}D_{2,2}\dots D_{k,2}D_{1,3}\dots$
Lorsque toutes les données ont été traitées, on écrit de la même manière les octets ajoutés pour la correction :
$E_{1,1}E_{2,1}\dots E_{k,1}E_{1,2}E_{2,2}\dots E_{k,2}E_{1,3}\dots$

Exemple pour un QR code version 5-H :
Pour aider à visualiser on peut écrire les blocs dans un tableau de la manière suivante :

Indice du bloc	Séquence des données					Octets de correction			
1	$D_{1,1}$	$D_{1,2}$	$\dots$	$D_{1,11}$		$E_{1,1}$	$E_{1,2}$	$\dots$	$E_{1,22}$
2	$D_{2,1}$	$D_{2,2}$	$\dots$	$D_{2,11}$		$E_{2,1}$	$E_{2,2}$	$\dots$	$E_{2,22}$
3	$D_{3,1}$	$D_{3,2}$	$\dots$	$D_{3,11}$	$D_{3,12}$	$E_{3,1}$	$E_{3,2}$	$\dots$	$E_{3,22}$
3	$D_{4,1}$	$D_{4,2}$	$\dots$	$D_{4,11}$	$D_{4,12}$	$E_{4,1}$	$E_{4,2}$	$\dots$	$E_{4,22}$

La séquence à écrire sera alors :
$D_{1,1}D_{2,1}D_{3,1}D_{4,1}D_{1,2}D_{2,2}D_{3,2}D_{4,2}\dots D_{1,11}D_{2,11}D_{3,11}D_{4,11}D_{3,12}D_{4,12}E_{1,1}E_{2,1}E_{3,1}E_{4,1}E_{1,2}\dots E_{1,22}E_{2,22}E_{3,22}E_{4,22}$

5.1.7 Écriture de la séquence finale sur le QR code

Par les procédés précédents on obtient une séquence finale de données suivies d'octets de correction des erreurs.
Nous allons maintenant voir comment écrire cette séquence sur un QR code.
On place les données par groupes de 8 bits de la manière suivante :
On commence en bas à droite en direction du haut.
Tant qu'on peut placer des mots dans notre direction on les place, et lorsque ce n'est plus possible on recommence en se décalant de 2 pixels vers la gauche et en recommençant dans la direction opposée à la direction précédente.
Ce qui donne sur un QR code version 1 le parcours suivant :

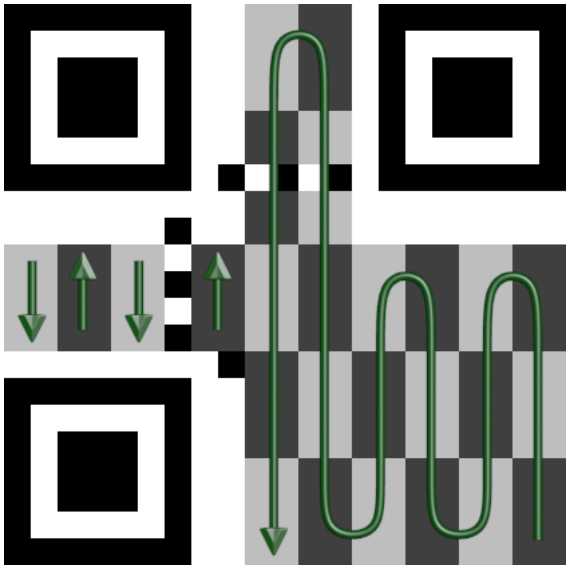


FIGURE 5.16 – *Parcours du QR code version 1*

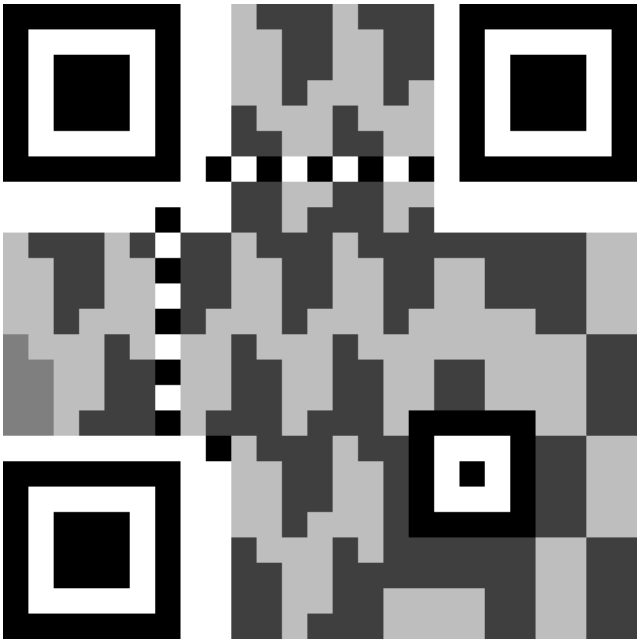
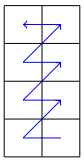
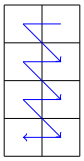


FIGURE 5.17 – *Placement sur un QR code version 2*

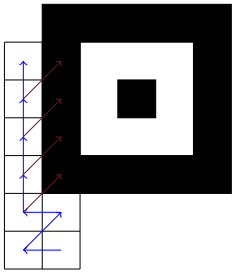
Maintenant étudions la manière de placer les bits dans une zone, premièrement dans un simple rectangle :
Lorsque la direction est vers le haut on place les bits de la manière suivante :



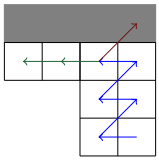
Et vers le bas de la manière suivante :



Pour généraliser la procédure, j'ai choisi de sélectionner le bit suivant par le procédé suivant :
On recherche dans la direction vers laquelle on avance le prochain bit en face de l'actuel et le prochain si on doit se décaler sur la colonne à côté.
Si les deux solutions sont à même distance on choisit le déplacement normal comme précédemment.
Sinon on choisit d'aller sur le bit le plus proche.
S'il n'existe pas de déplacement possible dans notre direction alors on change de direction et on se place sur le premier pixel le plus à droite (le plus haut si on avançait vers le haut et le plus bas si on avançait vers le bas).
Exemple de changement de sélection :



Exemple de changement de direction :



Exemples de QR codes pour les données 'Hello World !' :

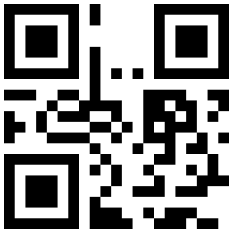


FIGURE 5.18 – *Version 1-L sans Masque*

FIGURE 5.19 – *Version 2-L sans Masque*

Il suffit en dernière étape d'appliquer un masque sur la partie qui concerne la séquence des données.
Il ne faut pas écrire sur la partie réservée au format ou sur les patterns d'alignement.

FIGURE 5.20 – *Version 2-L avec Masque 7*

5.2 Décodage

5.2.1 Récupération des données

Pour récupérer la séquence on suit le même parcours de bits que vu pour le codage. On découpe la séquence en deux parties : les données et les octets de correction des erreurs.
Et pour chaque bloc i (t_i, $t_{i,m}$, $t_{c,i}$) on reforme le polynôme associé m_i construit par les coefficients des données en plus haut degré et suivi de ceux de correction des données :

$$D_{i,1}x^{t_i-1}+D_{i,2}x^{t_i}+D_{i,3}x^{t_i-1}+\ldots+D_{i,t_{i,m}}x^{t_i-t_{i,m}}+E_{i,1}x^{t_i-t_{i,m}-1}+\ldots+E_{i,t_i-t_{i,m}-1}x+E_{i,t_i-t_{i,m}}$$

$$m_i(x) = \sum_{k=1}^{t_{i,m}} D_{i,k}x^{t_i-k} + \sum_{k=1}^{t_{i,m}-t_{i,m}} E_{i,k}x^{t_i-t_{i,m}-k}$$

5.2.2 Vérification du format

Dans un premier temps on récupère le format qu'on décode par un BCH (15, 10, 3) dont le corps fini est engendré par $x^4 + x + 1$.

La correction des erreurs et la récupération des données se fait comme pour les BCH.
La différence est que les coefficients de notre polynôme sont à valeurs dans $\mathbb{F}_{256}$ et non dans $\mathbb{F}_2$.
Nous noterons $m \in \mathbb{F}_{256}[X]$ le polynôme représentant pour un bloc(t, t_m, t_c) comme vu précédemment.
On suppose que le nombre d'erreurs est inférieur à t_c.
On notera $c = t - t_m$ le nombre d'octets de correction d'erreurs.

5.2.3 Calcul des syndromes

On calcule les c syndromes de la manière suivante :

$$s_i = m(\alpha^i)$$

Si pour tout $i \in [\![1, c]\!]$, $s_i = 0$, alors il n'y a pas d'erreurs à corriger sinon on lance le processus de correction des erreurs.

5.2.4 Polynôme localisateur d'erreurs

Pour déterminer le polynôme localisateur d'erreurs, on peut utiliser la méthode du pivot de Gauss comme pour les BCH.
Mais aussi pour les QR codes je choisi de calculer le polynôme localisateur l avec le formule de calcul de PGCD :

$$l(x)s(x) + v(x)x^{2t_c} = w(x)$$

Tel que w est le premier polynôme de $\mathbb{F}_{256}[X]$ où $deg(w) < t_c$.
On forme s le polynôme des syndromes de la même manière que pour les BCH.

5.2.5 Correction des erreurs

Pour trouver les erreurs on utilise la méthode de Chien Search.
Pour $i \in [\![0, t-1]\!]$, on notera α^{-i} l'inverse de α^i dans $\mathbb{F}_{256}$
Si $l(\alpha^{-i}) = 0$, alors on a une erreur à corriger au coefficient i
Pour obtenir la valeur de l'erreur j'ai utilisé l'algorithme de Forney.
C'est-à-dire qu'on construit le polynôme $\Omega \in \mathbb{F}_{256}[X]$ tel que :

$$\Omega(x) = s(x)l(x) \quad mod \quad x^{t_c}$$

On calcule la dérivée formelle de l que l'on nomme l'.
Pour rappel la dérivée formelle de l est donnée par :

$$l'(x) = \sum_{i=0}^{\lfloor \frac{deg(l)-1}{2} \rfloor} l_{2i+1} x^{2i}$$

On calcule notre erreur e_i telle que :

$$e_i = \frac{\alpha^i \Omega(\alpha^{-i})}{l'(\alpha^{-i})}$$

Ici la division sous-entend multiplication par l'inverse dans $\mathbb{F}_{256}$.
Il suffit maintenant d'additionner l'erreur e_i au coefficient d'indice i pour le corriger.

5.2.6 Exemple avec erreurs

FIGURE 5.21 – *QR code avec erreurs*

Ce QR code a malheureusement été partiellement effacé, on se propose de retrouver les données de ce QR code.
On récupère les formats suivants :
101010001000111
101011001000010
On les corrige à l'aide d'un BCH (15, 10, 3) engendré par $x^4 + x + 1$.
Les deux formats comportent des erreurs, il faut les corriger.
C'est-à-dire que le premier est erroné au coefficient à la position 9, ce qui donne :
101011001000111
Le second aux positions 0 et 2, ce qui donne aussi :
101011001000111
Le format est donc 10101
Autrement dit un QR code version 2-H avec pour masque 5.

On a donc un seul bloc sous la forme (44, 16, 14).
On extrait le polynôme suivant en tant que séquence de données :
$EAx^{43} + AEx^{42} + AAx^{41} + EAx^{40} + 93x^{39} + 19x^{38} + 31x^{37} + 26x^{36} + 97x^{35} + 46x^{34} + 96x^{33} + F6x^{32} + E2x^{31} + 2x^{30} + 10x^{29} + CBx^{27} + 7Dx^{26} + 59x^{25} + B1x^{24} + 7Ax^{23} + D0x^{22} + D7x^{21} + 8Bx^{20} + 8Cx^{19} + 9Ex^{18} + BBx^{17} + 9Dx^{16} + CDx^{15} + 2x^{14} + 37x^{13} + 5Fx^{12} + 82x^{11} + E3x^{10} + C0x^9 + CEx^8 + C5x^7 + 55x^6 + 80x^5 + 86x^4 + 9x^3 + 31x^2 + 92x + AE$

Pour le calcul des syndromes je les donne sous forme du polynôme des syndromes :
$s(x) = D4x^{26} + 39x^{25} + C3x^{24} + C3x^{23} + ADx^{22} + 6Fx^{21} + 3Ex^{20} + 4Ex^{19} + C5x^{18} + 48x^{17} + FEx^{16} + A9x^{15} + D5x^{14} + B6x^{13} + E7x^{12} + F4x^{11} + 98x^{10} + 98x^9 + FCx^8 + F6x^7 + 90x^6 + D1x^5 + A5x^4 + 61x^3 + 96x^2 + 2Ax + 1E$

On en déduit le polynôme localisateur :
$l(x) = 9Fx^{14} + 13x^{13} + F9x^{12} + 62x^{11} + 53x^{10} + 9Ex^9 + 51x^8 + 3Fx^7 + 91x^6 + Ax^5 + 3Cx^4 + 74x^3 + 8Ex^2 + E8x + 1$

On va maintenant préparer Ω et l' :
$\Omega(x) = 4Bx^{13} + E5x^{12} + AEx^{11} + 95x^{10} + 78x^9 + CCx^8 + 19x^7 + 6Cx^6 + 55x^5 + 9x^4 + 84x^3 + Bx^2 + 55x + 1E$

$l'(x) = 13x^{12} + 62x^{10} + 9Ex^8 + 3Fx^6 + Ax^4 + 74x^2 + E8$

On peut maintenant corriger les erreurs :

Position de l'erreur dans le polynôme	Valeur de l'erreur en hexadécimal
0	F0
1	5F
2	DA
3	77
4	68
5	AA
6	2
37	26
38	1F
39	A4
40	7D
41	EC
42	7A
43	AA

On a corrigé 14 erreurs, ce qui était de plus la capacité d'erreur maximale corrigeable par le QR code.
On peut maintenant lire le message 'Disparition !'

Chapitre 6

Manuel d'utilisation

6.1 Compilation

On peut compiler 3 exécutables :
- code_correcteur : fonctionnalités sur les polynômes et codes BCH
- qr_code : création et décodage des QR codes
- qr_code_anim : création et décodage des QR codes avec animation lors de lecture/écriture des données

Pour cela si *executable* est l'exécutable choisi il suffit de faire :

```
make executable
```

Pour effacer les exécutables, il existe aussi une option clean dans le Makefile.
Les exécutables des QR codes nécessitent l'installation de la bibliothèque SDL 1.2
Exemple de commandes pour installer SDL sur Ubuntu :

```
sudo apt-get install libsdl-gfx1.2-dev libsdl1.2-dev libsdl-mixer1.2-dev
sudo apt-get install libsdl-ttf2.0-dev libglib2.0-dev libxml2-dev
sudo apt-get install libsdl-image1.2-dev
```

Chaque exécutable possède une option -h ou - -help pour obtenir une aide rapide sur les commandes et arguments.

6.2 Recherche de polynômes irréductibles dans $\mathbb{F}_2[X]$

Les polynômes sont écris sous forme binaire.
Par exemple $x^4 + x + 1$ sera écrit sous la forme 10011.

6.2.1 Crible d'Eratosthène

On lance la recherche de polynômes par le crible d'Eratosthène de la manière suivante :

```
./code_correcteur --simple-generate
./code_correcteur -s
./code_correcteur -s [DEGRE]
```

Par défaut la recherche se fait jusqu'au degré 16 mais il est possible de préciser un autre degré auquel s'arrêter dans la recherche.
Les polynômes trouvés par la recherche sont ajoutés dans le dossier 'media/f2_primes'.

6.2.2 Recherche par aléatoire

Pour la recherche de polynômes irréductibles par génération aléatoire, on se donne une fourchette de degrés dans laquelle on va construire un polynôme aléatoirement dans le but de l'étudier.

```
./code_correcteur --generate [DEGRE]
./code_correcteur -g [DEGRE]
./code_correcteur -g [DEGRE_MIN] [DEGRE_MAX]
```

On peut directement tomber sur un polynôme irréductible ce qui reste rare à haut degré. Pour cela si un polynôme est factorisable, on va rechercher récursivement si ses facteurs sont irréductibles avec le même procédé.
Il est donc conseillé de prendre un fourchette au dessus du degré qu'on recherche.

6.2.3 Lister les polynômes irréductibles

On peut lister tous les polynômes irréductibles trouvés pour un certain degré, on utilise la commande suivante :

```
./code_correcteur --list [DEGRE]
./code_correcteur -l [DEGRE]
```

On peut aussi demander un seul polynôme irréductible aléatoirement dans la liste avec la commande suivante :

```
./code_correcteur --random [DEGRE]
./code_correcteur -r [DEGRE]
```

6.2.4 Berlekamp

Pour obtenir un factorisation par l'algorithme de Berlekamp, on peut utiliser la commande suivante :

```
./code_correcteur --berlekamp [POLYNOME]
./code_correcteur -b [POLYNOME]
```

On écrit le polynôme sous sa forme binaire.

6.3 Calculatrice

6.3.1 Dans $\mathbb{F}_2[X]$

On lance la calculatrice dans $\mathbb{F}_2[X]$ par la commande suivante :

```
./code_correcteur --calculator
./code_correcteur -c
```

On représente les polynômes par leur forme binaire.
Le résultat est affiché sous la forme : '(degré) : polynôme en binaire'
Il est aussi possible d'écrire un polynôme sous une forme hexadécimale en ajoutant 'h' devant l'écriture du polynôme.
Exemple : $hA = 1010 = x^3 + x$

Addition/Soustraction

Pour l'addition et la soustraction qui sont ici le 'xor' on utilise les symboles '+' ou '-'.
Exemple d'addition de $x^2 + x + 1$ et x :

```
111 + 10
(2) : 101
```

On peut mettre faire des additions successives de la manière suivante :

```
111 + 10 + 11001 + 10011 + 11
(3) : 1100
```

Multiplication

La multiplication s'effectue par le symbole '*'.
Exemple de multiplication de $x^2 + x + 1$ et $x + 1$:

```
111 * 11
(3) : 1001
```

De la même manière que l'addition on peut faire des multiplications successives :

```
111 * 11 * 10011 * 1011
(10) : 10111000101
```

Par contre pour mélanger addition et multiplication il faut ajouter des parenthèses :

```
111 * 11 * (111 + (11 * 11) + 10011) * 1011
(7) : 10011001
```

Pour la multiplication on peut aussi les effectuer modulo un polynôme en mettant à la fin le polynôme avec lequel on effectue le modulo.
On peut utiliser le modulo dans le cas de deux opérandes.
Exemple de calcul modulo 10011 :

```
(11 * 11) * 1011 [10011]
(0) : 1
```

Division

L'opération de division ici est la récupération du quotient de la division euclidienne.
On utilise l'opérateur '/' :

```
101 / 11
(1) : 11
111 / 11
(1) : 10
```

Modulo

Le modulo est ici le reste de la division euclidienne, avec l'opérateur '%' :

```
1011 % 111
(1) : 10
101 % 11
(0) : 0
```

PGCD

Pour le calcul du PGCD, on utilise l'opérateur '^'.
De plus on nous donne u et v tels qu'ils vérifient l'identité de Bezout.

```
1001 ^ 101
u : 1
v : 10
(1) : 11
```

Inverse

On désigne 'i' l'opérateur qui pour $x \quad i \quad y$ nous donne l'inverse de x par y.

```
1011 i 10011
(2) : 101
```

Composition

Pour la composition si p et q sont deux polynôme, j'appelle composition l'évaluation de $p(q(x))$.
On notera cette composition par l'opérateur '<' tel que $p(q(x))$ se calcule $p < q$

```
111 < 1000
(6) : 1001001
1000 < 111
(6) : 1101011
10 < 111
(2) : 111
100 < 11
(2) : 101
1010101 < 11 [10011]
(3) : 1100
```

Test de primalité

On peut tester si un polynôme est irréductible par l'opérateur unaire ' ? '

```
?111
Irreductible
(2) : 111
?101
Produit de polynomes
(2) : 101
```

Test de primitivité

Pour tester si un polynôme irréductible est primitif lorsque α est un élément primitif représentant x, on peut utiliser l'opérateur unaire 'p'.

```
p10011
Primitif
(4) : 10011
p11111
Non Primitif
(4) : 11111
```

6.3.2 Dans $\mathbb{F}_{2^q}[X]$ par construction

Ici on choisi un polynôme p irréductible écrit en binaire pour construire les corps $\mathbb{F}_{2^q} = \mathbb{F}_2[X]/p(X)$.
Ce qui permet à notre calculatrice de travailler dans $\mathbb{F}_{2^q}[X]$.

```
./code_correcteur --calculator [POLYNOME]
./code_correcteur -c [POLYNOME]
```

Pour écrire un polynôme on écrit le coefficient suivi de 'X' puis le degré du monôme (aucun degré pour x^1 et simplement le coefficient pour x^0).
Pour considérer la suite de monômes comme une seule entité on sépare les monômes par '.'
Exemple $110x^2 + x + 11$ s'écrit :

```
1101 1111 X2 . X . 11
```

On peut aussi prendre la notation hexadécimale pour les coefficients :

```
hDF X2 . h1 X . h3
```

Tout comme dans $\mathbb{F}_2[X]$ avec les mêmes opérateurs, les opérations suivantes sont définies :

- '+' Addition
- '*' Multiplication
- '/' Division (Quotient)
- '%' Modulo
- '^' PGCD
- '<' Composition

6.4 BCH

On peut lancer des tests de simulation de BCH.
Si on précise un polynôme les tests aléatoires se font par le BCH engendré par ce polynôme.
Sinon un polynôme primitif est choisi aléatoirement à chaque essai.

```
./code_correcteur --test-bch
./code_correcteur -t-bch
./code_correcteur --test-bch [POLYNOME]
```

6.4.1 Codage

```
./code_correcteur -bch [NOMBRE DE CORRECTIONS] [POLYNOME PRIMITIF] -c [
    POLYNOME MESSAGE]
./code_correcteur -bch [NOMBRE DE CORRECTIONS] [POLYNOME PRIMITIF] --code [
    POLYNOME MESSAGE]
```

On donne la capacité de correction, le polynôme primitif utilisé sous forme binaire, puis le message sous forme binaire.

```
./code_correcteur -bch 3 10011 -c 10111
Message cod   : 101110000101001
```

6.4.2 Décodage

```
./code_correcteur -bch [NOMBRE DE CORRECTIONS] [POLYNOME PRIMITIF] -d [
    POLYNOME MESSAGE]
./code_correcteur -bch [NOMBRE DE CORRECTIONS] [POLYNOME PRIMITIF] --decode
     [POLYNOME MESSAGE]
```

On donne les mêmes informations que pour le codage pour construire le bch puis le message à décoder.

```
./code_correcteur -bch 3 10011 -d 111110000101111
Correction erreur en 13
Correction erreur en 2
Correction erreur en 1
Message d cod   : 10111
```

6.5 QR Codes

6.5.1 Codage

Pour créer un QR code, on utilise l'exécutable de la manière suivante :

```
./qr_code [VERSION] [TAUX DE CORRECTION][MASQUE] [MESSAGE]
./qr_code [VERSION] [TAUX DE CORRECTION][MASQUE] [MESSAGE] -s [CHEMIN DE SAUVEGARDE]
./qr_code [VERSION] [TAUX DE CORRECTION][MASQUE] [MESSAGE] -s [CHEMIN DE SAUVEGARDE] [ZOOM]
./qr_code_anim [VERSION] [TAUX DE CORRECTION][MASQUE] [MESSAGE] -s [CHEMIN DE SAUVEGARDE] [ZOOM]
```

Pour le moment seulement les versions 1 et 2 du QR code sont implémentées.
Pour le taux de correction, on rappelle les différents taux :

Taux de correction	Pourcentage d'erreurs corrigeables
L	7
M	15
Q	25
H	30

Le masque est un entier entre 0 et 7.
On ne laisse pas d'espace entre le taux et le masque.
Si on veut sauvegarder l'image générée il est possible d'indiquer un format de sauvegarde au format '.bmp'.
Le Zoom est la taille que doit faire un pixel réel de données sur l'image finale.

```
./qr_code 2 H5 "Hello World !"
./qr_code 2 H5 "Hello World !" -s "media/hello_world.bmp"
./qr_code 2 H5 "Hello World !" -s "media/hello_world.bmp" 20
./qr_code_anim 2 H5 "Hello World !"
```

6.5.2 Décodage

Pour lire un QR code existant il suffit simplement d'indiquer le chemin de l'image à ouvrir.
L'image peut être à un format standard autre que '.bmp' comme '.png' par exemple.

```
./qr_code -o [CHEMIN DE L'IMAGE]
./qr_code --open [CHEMIN DE L'IMAGE]
./qr_code_anim --open [CHEMIN DE L'IMAGE]
```

```
./qr_code -o "media/hello_world.bmp"
./qr_code_anim -o "media/hello_world.bmp"
```

Chapitre 7

Bilan du projet

7.1 Perspectives

Pour continuer d'approfondir ce projet, je pense intéressant d'étudier les idées suivantes :

- Implémenter les fonctionnalités liées aux polynômes irréductibles dans $\mathbb{F}_2[X]$ à $\mathbb{F}_{2^q}[X]$ en rendant toutes les fonctionnalités génériques dans les corps finis
- Étudier des méthodes plus efficaces pour construire et trouver tous les polynômes irréductibles, par exemple trouver un morphisme liant les mots de Lyndon aux polynômes irréductibles de $\mathbb{F}_2[X]$
- Mettre en place un mécanisme pour transformer un QR déformé sur une image en un QR code plat facile à décoder
- Créer un module permettant la détection de QR code sur une image
- Implémenter plus de versions des QR codes, pour cela mieux gérer les blocs car le programme est principalement adapté à gérer et corriger un bloc
- Implémenter les QR codes colorés (2013-2014) : High Capacity Colored 2-Dimensional (HCC2D) Code

7.2 Conclusion

Les moments d'autonomie m'ont permis de chercher de moi même à approfondir et comprendre, mais m'ont aussi fait perdre plus de temps car j'ai avancé beaucoup plus vite et mieux compris après des explications par Messieurs Thibon et Zipstein, ce qui m'aurait sûrement permis de pousser le projet plus loin.
Cependant ce projet était très instructif et intéressant, il m'a aussi bien permis de m'initier à la cryptographie par l'étude et l'implémentation des codes correcteurs que permis d'approfondir des notions Mathématiques telles que l'arithmétique modulaire, ici par la notion de corps finis avec les polynômes de $\mathbb{F}_2[X]$.
De plus nous nous sommes aussi intéressés à la construction et le décodage de QR codes qui sont une utilisation contemporaine des codes correcteurs.

Chapitre 8

Bibliographie

— PRETZEL Oliver ‘Error-Correcting Codes and Finite Fields‘, Clarendon Press - OXFORD 1992
— FORICHON Virginie ‘CODES CORRECTEURS D’ERREURS : CONSTRUCTION ET EXEMPLES‘, Mémoire, Université Paris-Est-Marne-La-Vallée - 2002-2003
— Spécification Norme QR codes ISO/IEC 18004 :2006(E) second edition 2006-09-01
— Page Wikipédia sur les corps finis : `https://fr.wikipedia.org/wiki/Corps_fini`
— Page Wikipédia sur les codes de Reed-Solomon : `https://en.wikipedia.org/wiki/Reed%E2%80%93Solomon_error_correction`
— Page Wikipédia Forney algorithm : `https://en.wikipedia.org/wiki/Forney_algorithm`