TP 3 : Codes correcteurs d'erreurs : CRC

Dans ce TP, vous allez implanter les fonctions de Contrôle de Redondance Cyclique (CRC) en Python (§ 2). Au préalable, il est nécessaire d'implanter des opérations arithmétiques sur des polynômes sur $\mathbf{Z}/2\mathbf{Z}$ (§ 1).

1 Opérations sur des polynômes

Les polynômes sur $\mathbf{Z}/2\mathbf{Z}$ seront représentés comme des listes de nombres 0, 1. Pour avoir une correspondance directe entre l'exposant du monôme et la position dans la liste, le polynôme $P(X) = a_nX^n + a_{n-1}X^{n-1} + \cdots + a_2X^2 + a_1X + a_0$ sera représenté par la liste $[a_0, a_1, \ldots, a_n]$, par exemple $X^3 + X^2 + 1$ par $[1, 0, 1, 1]$. Le cours a utilisé la représentation inverse, à savoir $[1, 1, 0, 1]$.

Dans cette section, nous n'imposons pas de "forme normale" des représentations des polynômes, surtout en ce qui concerne des 0 superflus à droite de la représentation. Ainsi, le polynôme $X^3 + X^2 + 1 = 0 * X^5 + 0 * x^4 + 1 * X^3 + 1 * X^2 + 0 * X + 1$ peut être représenté par $[1, 0, 1, 1]$ ou par $[1, 0, 1, 1, 0, 0]$. En particulier, le polynôme 0 a plusieurs représentations : [ ], [0], [0, 0] etc.

Vos fonctions doivent donc être correctes pour toutes les représentations d'un polynôme. Il faut surtout être prudent avec des comparaisons. Par exemple, $[1, 0, 1, 1]$ et $[1, 0, 1, 1, 0, 0]$ représentent le même polynôme, mais le test `[1, 0, 1, 1] == [1, 0, 1, 1, 0, 0]` en Python donne le résultat `False`.

Exercice 1 Pour commencer, écrivez la fonction `add_mod2` qui additionne deux nombres (0 ou 1) modulo 2.

Exercice 2 Écrivez la fonction `deg` qui calcule le degré d'un polynôme. D'après les remarques précédentes, le degré ne correspond pas forcément à la longueur de la représentation.

Exercice 3 Écrivez la fonction `coeff` qui calcule le coefficient dominant d'un polynôme. Sur $\mathbf{Z}/2\mathbf{Z}$, celui-ci est toujours 1 sauf si le polynôme est 0.

Exercice 4 Écrivez la fonction `is_null_poly` qui teste si un polynôme `p` est équivalent au polynôme 0. Cette fonction renvoie un Booléen. D'après les remarques précédentes, une simple comparaison de la forme `p == [0]` ne suffit pas.

Exemples d'utilisation de ces fonctions :

```
>>> deg([0,1,1])
2
>>> deg([0,1,1,0])
2
>>> deg([])
0
>>> coeff([0,1,1])
1
>>> coeff([0,1,1,0])
1
>>> coeff([0, 0])
0
>>> coeff([])
```

```
0
>>> is_null_poly([])
True
>>> is_null_poly([0, 0])
True
>>> is_null_poly([0, 1, 0])
False
```

Exercice 5 Écrivez la fonction `add_poly` qui additionne deux polynômes.

Exercice 6 Écrivez la fonction `diff_poly` qui calcule la différence entre deux polynômes. Comme remarqué en cours, les fonctions d'addition et soustraction sont identiques sur $\mathbf{Z}/2\mathbf{Z}$ (et nous espérons que vous avez appelé la fonction `add_poly` au lieu d'en copier le code). Nous introduisons cette fonction uniquement pour des raisons de lisibilité.

Exercice 7 Écrivez la fonction `mult_poly` qui multiplie deux polynômes. Il est très utile d'introduire des fonctions auxiliaires (à vous de les identifier), par exemple pour calculer le produit d'un polynôme avec un monôme, de la forme $P(X) * b_k X^k$.

Exercice 8 Écrivez la fonction `eucl_poly` qui effectue la division Euclidienne de deux polynômes, et qui renvoie deux polynômes, à savoir le quotient et le reste. Vous pouvez copier presque directement l'algorithme donné sur les transparents du cours, avec de petites adaptations syntaxiques, et en utilisant les fonctions `add_poly`, `mult_poly` etc. écrites précédemment. Nous supposons que le diviseur n'est pas 0, il n'est donc pas nécessaire de vérifier cette propriété dans votre fonction `eucl_poly`.

Note (couples en Python) : cette fonction renvoie un *couple* de polynômes (et non une liste de polynômes). Un couple est formé par deux expressions entre parenthèses, séparées par une virgule. Vous pouvez obtenir sélectionner le premier (resp. deuxième) composant d'un couple comme dans le cas des listes, par exemple : `(10, 11)[0]` donne `10`. Vous pouvez aussi affecter un couple à deux variables en parallèle, par exemple : `(a, b) = (10, 11)` affecte `10` à la variable `a` et `11` à `b`.

Exercice 9 A l'aide de la fonction `eucl_poly`, il est très facile d'écrire les fonctions `div_poly` et `mod_poly` qui calculent uniquement le quotient et le reste de la division.

Exemples d'utilisation de ces fonctions :

```
>>> add_poly([0,1,1], [1,0,0,1])
[1, 1, 1, 1]
>>> add_poly([0,1,1], [1,0,1,1])
[1, 1, 0, 1]
>>> add_poly([0,1,1,0], [1,0,1,1,0])
[1, 1, 0, 1, 0]
>>> mult_poly([0,1,1], [1,0,1,1])
[0, 1, 1, 1, 0, 1]
>>> eucl_poly([0,1,1,0,1], [1,1])
([1, 0, 1, 1], [1, 0, 0, 0, 0])
>>> (p, r) = eucl_poly([0,1,1,0,1], [1,1])
>>> p
[1, 0, 1, 1]
>>> r
[1, 0, 0, 0, 0]
>>> mod_poly([0,1,1,0,1], [1,1])
[1, 0, 0, 0, 0]
```

2 Codage et décodage de messages

Sur la base des fonctions de la § 1, vous pouvez écrire des fonctions de codage et décodage de polynômes selon la méthode CRC.

Contrairement au traitement "mathématique" des polynômes de la § 1, qui admettait plusieurs représentations équivalentes d'un polynôme, nous adoptons une approche plus "informatique" ici et distinguons deux séquences de 0 et 1 même si elles représentent le même polynôme. Ainsi, dans un contexte informatique, les séquences `[1, 0, 0, 0]` et `[1, 0]` sont différentes, même si elles représentent le même polynôme, et il ne serait pas acceptable de coder la séquence `[1, 0, 0, 0]` et de la décoder comme `[1, 0]`.

Exercice 10 Écrivez la fonction `pad` qui prend deux arguments (une représentation de polynôme et un nombre) et rempli la représentation de polynôme avec des 0 à droite.

Exercice 11 Écrivez la fonction `cod` qui prend un message, un polynôme générateur et la taille souhaitée du message codé, et qui calcule le message codé (avec la taille souhaitée).

Exercice 12 Écrivez la fonction `decod` qui prend un message codé, un polynôme générateur et la taille souhaitée du message décodé, et qui calcule un couple : le message décodé (avec la taille souhaitée), et un Booléen qui inique si le décodage était possible.

Exemples d'utilisation de ces fonctions :

```
>>> cod([0,1,1,0,1], [1,0,1], 7)
[0, 1, 0, 1, 1, 0, 1]
>>> decod([0, 1, 0, 1, 1, 0, 1], [1,0,1], 5)
([0, 1, 1, 0, 1], True)
>>> decod([0, 1, 0, 1, 0, 1, 1], [1,0,1], 5)
([0, 1, 0, 1, 1], False)
>>> decod([1, 0, 1, 0, 1, 0, 1], [1,0,1], 5)
([1, 0, 1, 0, 1], True)
>>> cod([0,1,0,1,0,0], [1,0,1], 7)
[0, 0, 0, 1, 0, 1, 0, 0]
>>> decod([0, 0, 0, 1, 0, 1, 0, 0], [1,0,1], 6)
([0, 1, 0, 1, 0, 0], True)
```