

Codes correcteurs

TP1 : Introduction

Ce TP sera implémenté à l'intérieur d'un seul et même fichier C afin de former un programme complet.

1. Vecteurs binaires

Une représentation simple de vecteur est le tableau d'entiers. Les exercices suivants visent à implanter les fonctionnalités nécessaires pour l'utilisation de vecteurs.

Exercice 1.1 : Définir un type VECTEUR comme étant un `int*`. (voir [typedef](#))

Exercice 1.2 : Ecrire une fonction `int pow2(unsigned int n)` qui retourne la valeur 2^n .

Exercice 1.3 : Ecrire une fonction `VECTEUR vecteur_vide(unsigned int n)` qui crée un vecteur binaire de n bits sous forme d'un tableau de n valeurs. Les valeurs du vecteur créé sont initialisées à 0.

Exercice 1.4 : Ecrire une fonction `void affiche_vecteur(VECTEUR v, unsigned int n)` qui affiche sur la sortie standard le vecteur v . Le vecteur est affiché dans l'ordre croissant de ses composantes :

$$(v_1, \dots, v_i, \dots, v_n)$$

Avec $v_i = v[i-1]$

Exercice 1.5 : Ecrire une fonction `VECTEUR vecteur(unsigned int n, unsigned int valeur)` qui crée un vecteur binaire de n bits représentant la valeur passée en paramètres. Si la valeur ne peut être codée sur les n bits, alors le vecteur nul (NULL) est retourné. Les composantes du vecteur créé sont représentées telles la valeur i du vecteur retourné correspond au coefficient de 2^i dans la représentation binaire du nombre. Par exemple, si la valeur est décomposée en binaire sous la forme :

$$a \times 2^0 + b \times 2^1 + c \times 2^2 + d \times 2^3$$

Le vecteur v retourné est tel que $v[0] = a$, $v[1] = b$, $v[2] = c$ et $v[3] = d$.

Exercice 1.6 : Soit v un vecteur binaire de taille n tel que $v = (v_1, \dots, v_i, \dots, v_n)$. La valeur décimale val correspondant à la représentation binaire portée par v est donnée par :

$$val = \sum_{i=1}^n v_i 2^{i-1}$$

Ecrire une fonction `int valeur(VECTEUR v, unsigned int n)` qui retourne la valeur de la représentation binaire donnée par le vecteur v de taille n . Si le vecteur v est vide, -1 est retourné.

2. Mots

Les codes correcteurs reposent sur la notion de mot. Un mot peut être vu comme un vecteur binaire avec des fonctionnalités permettant entre autres de comparer des mots entre eux.

Exercice 2.1 : Ecrire une fonction `VECTEUR* mots(unsigned int k)` qui génère sous forme de tableau de vecteurs tous les 2^k vecteurs binaires possibles de taille k . Si k est négatif ou nul alors `NULL` est retourné.

Par exemple, pour une taille de 3, l'ensemble des 2^3 vecteurs possibles est :

$$\{(0,0,0), (0,0,1), (0,1,0), (0,1,1), (1,0,0), (1,0,1), (1,1,0), (1,1,1)\}$$

La fonction `VECTEUR vecteur(unsigned int n, unsigned int valeur)` peut être appelée pour créer les vecteurs.

Exercice 2.2 : Sachant que le poids d'un vecteur binaire est le nombre de ses composantes égales à 1. Ecrire une fonction `unsigned int poids(VECTEUR v, int n)` qui renvoie le poids du vecteur binaire v de taille n passé en paramètre. Si le vecteur est vide alors \emptyset est renvoyé.

Exercice 2.3 : Soient deux vecteurs u et v de taille n . Le vecteur différence, noté d , de u et v est lui-même un vecteur de taille n dont la composante d_i vaut 0 si les composantes u_i et v_i sont égales et 1 vaut si les composantes u_i et v_i sont différentes. Plus formellement, dans le cadre de vecteurs binaires :

$$d = (d_i), 1 \leq i \leq n \text{ avec } d_i = u_i \oplus v_i$$

Où \oplus est le ou exclusif.

Ecrire une fonction `VECTEUR diff(VECTEUR u, VECTEUR v, int n)` qui retourne le vecteur binaire différence entre les vecteurs binaires u et v de taille n passés en paramètres. Si l'un des deux vecteurs est vide (ou de taille 0), alors `NULL` est retourné.

Exercice 2.4 : Ecrire une fonction `unsigned int hamming(VECTEUR u, VECTEUR v, int n)` qui calcule la distance de Hamming entre les vecteurs binaires u et v , tous deux de taille n . Si l'un des deux vecteurs est vide (ou de taille 0), alors \emptyset est retourné. Les fonctions `poids` et `diff` peuvent être utilisées.

Pour rappel, la distance de Hamming est le nombre de composantes différentes entre 2 vecteurs binaires.

3. Codage

L'encodage via des codes correcteurs linéaires repose sur l'utilisation d'une matrice génératrice. Cette matrice peut être représentée par un tableau dont chaque ligne est un vecteur.

Exercice 3.1 : Définir un type `MATRICE` comme étant un `int**`.

Exercice 3.2 : Ecrire une fonction `void affiche_matrice(MATRICE mat, unsigned int l, unsigned int c)` qui affiche sur la sortie standard la matrice représentée par le tableau à deux dimensions mat ayant l lignes et c colonnes. L'affichage sera de la forme :

```
[ mat[0][0] ... mat[0][j] ... mat[0][c - 1] ]
[      ...      ... mat[i][j] ...      ...      ]
[ mat[l-1][0] ... mat[l-1][j] ... mat[l-1][c-1] ]
```

Exercice 3.3 : Soit G une matrice génératrice de taille $k \times n$. définie telle que :

$$G = \begin{bmatrix} g_{1,1} & \dots & g_{1,n} \\ \vdots & g_{i,j} & \vdots \\ g_{k,1} & \dots & g_{k,n} \end{bmatrix}$$

Le vecteur c résultant de l'encodage d'un vecteur $v = (v_1, \dots, v_k)$ de dimension k par la matrice G est tel que :

$$c = vG = (v_1, \dots, v_i, \dots, v_k) \times G = (c_1, \dots, c_j, \dots, c_n), \text{ avec } c_j = \sum_{i=1}^k v_i g_{i,j}$$

Où \times représente le produit matriciel et où l'opérateur de ou exclusif \oplus est l'opérateur d'addition.

Ecrire une fonction `VECTEUR encode(MATRICE g, VECTEUR v, unsigned int k, unsigned int n)` qui encode le vecteur binaire v de taille k à l'aide de la matrice génératrice g représentant le code $C(n, k)$.

Exercice 3.4 : Utiliser la fonction `encode` implantée en 3.3 et produire l'ensemble des mots du code généré par la matrice G définie telle que :

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

L'ensemble des mots d'un code $C(n, k)$ est l'ensemble des vecteurs de taille n obtenus en encodant tous les vecteurs v de dimension k . Vous pouvez utiliser la fonction `VECTEURS* mots(unsigned int k)` pour gérer l'ensemble des mots de taille 4 nécessaire.

Exercice 3.5 : Ecrire la fonction `unsigned int dist_min(VECTEUR* vecteurs, unsigned int n, unsigned int nb_vect)` qui retourne la distance de Hamming minimale entre deux vecteurs distincts de l'ensemble de `nb_vect` vecteurs passé en paramètres. Si l'ensemble des vecteurs est vide ou inférieur à 2 alors 0 est retourné.

Exercice 3.6 : Sachant que la capacité de décodage e d'un code est donnée par la formule :

$$e = \left\lfloor \frac{d-1}{2} \right\rfloor$$

Où d est la distance minimale entre les mots du code.

Calculer la valeur de e pour le code dont la matrice génératrice est donnée en 3.4.

4. Contrôle

Le contrôle de vecteurs repose sur la notion de matrice de contrôle et le calcul de syndromes. Un syndrome étant un vecteur binaire.

Exercice 4.1 : Définir un type SYNDROME comme étant un int*.

Exercice 4.2 : Ecrire une fonction SYNDROME syndrome(MATRICE h, VECTEUR c, unsigned int k, unsigned int n) qui retourne le syndrome du vecteur c par la matrice h.

Pour rappel, soit H une matrice de contrôle de taille $n-k, n$. Le syndrome s d'un vecteur c est tel que :

$$s = cH^t$$

Où H^t est la transposée de la matrice H.

Avec \times représentant le produit matriciel et la matrice H telle que :

$$H = \begin{bmatrix} h_{1,1} & \cdots & h_{1,j} & \cdots & h_{1,n} \\ \vdots & & h_{i,j} & & \vdots \\ h_{n-k,1} & \cdots & h_{n-k,j} & \cdots & h_{n-k,n} \end{bmatrix}$$

Et dont la transposée est :

$$H^t = \begin{bmatrix} h_{1,1} & \cdots & h_{n-k,1} \\ \vdots & & \vdots \\ h_{1,j} & h_{i,j} & h_{n-k,j} \\ \vdots & & \vdots \\ h_{1,n} & \cdots & h_{n-k,n} \end{bmatrix}$$

Le développement du calcul précédent s'écrit :

$$s = (s_1, \dots, s_i, \dots, s_{n-k}), \text{ avec } s_i = \sum_{j=1}^n c_j h_{i,j}$$

Exercice 4.3 : En utilisant la fonction syndrome implantée en 3.1, calculer l'ensemble des syndromes issus des vecteurs de dimension 7 au travers de la matrice de contrôle H telle que :

$$H = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

Que peut-on déduire de l'étude des syndromes par rapport aux mots du code ?

5. Bruitage

Exercice 4.1 : Ecrire une fonction `VECTEUR bruité(VECTEUR v, unsigned int n, unsigned int b)` qui prends en paramètre un vecteur `v` de taille `n` et inverse le bit situé à l'indice `b` ($0 \leq b < n$). Le vecteur en entrée n'est pas modifié, une copie est retournée. Si le vecteur d'entrée est nul ou si l'indice `b` est invalide, alors `NULL` est retourné.

Exercice 4.2 : Tester la fonction `bruité` avec les vecteurs suivants (l'indice 0 est à gauche) :

Vecteur	Indice à modifier	Résultat attendu
(0, 0, 0, 0, 0, 0, 0)	0	(1, 0, 0, 0, 0, 0, 0)
(0, 1, 0, 0, 1, 1, 1)	1	(0, 0, 0, 0, 1, 1, 1)
(0, 0, 1, 0, 1, 1, 0)	2	(0, 0, 0, 0, 1, 1, 0)
(0, 1, 1, 0, 0, 0, 1)	3	(0, 1, 1, 1, 0, 0, 1)
(1, 1, 1, 0, 1, 0, 0)	4	(1, 1, 1, 0, 0, 0, 0)
(1, 0, 0, 1, 1, 1, 0)	5	(1, 0, 0, 1, 1, 0, 0)
(1, 0, 1, 1, 0, 0, 0)	6	(1, 0, 1, 1, 0, 0, 1)

Votre programme affichera ses résultats sous la forme :

<vecteur original> - <syndrome original> -> <vecteur bruité> - syndrome

5. Correction et décodage

Exercice 5.1 : A partir des résultats de l'exercice 4.2, et en prenant en compte la matrice de contrôle `H` de l'exercice 4.3, faire apparaître une corrélation entre un syndrome non nul et la position du bit erroné dans un mot transmis. **Indice :** trouver tout d'abord une corrélation entre les syndromes et la matrice `H`.

Exercice 5.2 : Ecrire une fonction `int indice_colonne(SYNDROME s, MATRICE h, unsigned int k, unsigned int n)` qui retourne pour la matrice `h` de taille $n-k$, `n` l'indice de sa colonne égale au syndrome `s` de taille $n-k$ passé en paramètre. Si aucune colonne de la matrice ne correspond ou si l'une des 2 variables `s` ou `h` est nulle alors `-1` est renvoyé. **Attention :** Prendre garde à l'ordre des indices entre la matrice et le vecteur.

Exercice 5.4 : Ecrire une fonction `VECTEUR corrige(VECTEUR v, MATRICE h, unsigned int k, unsigned int n)` qui corrige le vecteur `v` de taille `n` selon le code correcteur de matrice de contrôle `h` de taille $n-k$, `n`. Si le vecteur `v` n'est pas entaché d'erreur, il est retourné sans modification. Cette fonction ne doit pas modifier directement `v`.

Exercice 5.5 : Ecrire une fonction `VECTEUR decode(VECTEUR v, unsigned int k, unsigned int n)` qui décode le vecteur `v`. On considère que le vecteur `v` est un mot du code (mot corrigé d'erreur).

La correction se trouve ici : <http://www.seinturier.fr/cours/utln/info/licence/l3/i61/tp1.c>