Test d'auto-évaluation pour le Contrôle Terminal

BEGIN SOLUTION

Version pour enseignants

END SOLUTION

1 Inégalité de Kraft

Exercice 1 Supposez que vous voulez construire un système de codage préfixe binaire comprenant 16 codes de taille n et 32 codes de taille $n+1$. Quel est le n minimal qui permet de satisfaire ces contraintes ?

BEGIN SOLUTION

Selon l'inégalité de Kraft, il faut résoudre l'inégalité $16 * 2^{-n} + 32 * 2^{-(n+1)} \leq 1$. La solution est $n = 5$.

END SOLUTION

2 Algorithme de Huffman et LZW

Les exercices suivants ont pour but de comparer les algorithmes de Huffman et LZW, et enfin, de les combiner. Les exercices ont en commun qu'on travaille avec un alphabet $A = \{a, b, c\}$, et qu'on s'intéresse à coder un texte à 25 caractères $t =$`ababcabcaabcaabcaabcaabca` sur cet alphabet.

Exercice 2 Une première question très simple : Si nous codons les caractères de l'alphabet A en binaire (a par `00`, b par `01`, c par `10`), quel sera le nombre de bits requis ? Nous ne demandons pas d'effectuer le codage.

BEGIN SOLUTION

25*2=50 bits.

END SOLUTION

Exercice 3 Dans cet exercice, il s'agit de coder le texte t par un simple codage à l'aide de l'algorithme de Huffman :

1. Partez directement du texte t pour calculer la distribution de fréquence des caractères a, b, c dans le texte.
2. Sur cette base, construisez l'arbre de Huffman et attribuez un code binaire optimal à chacun des caractères de l'alphabet A. Quel est la taille (en bits) de t sous ce codage ?
3. Calculez l'entropie du texte t, et comparez la taille effective du codage que vous venez de concevoir avec l'optimum que vous pouvez atteindre.

BEGIN SOLUTION

1. Nombre d'occurrences : dans le tableau suivant ; distribution de fréquence : diviser par 25 (longueur du texte).
2. Code comme indiqué dans le tableau.

Caractère	a	b	c
# occur.	12	7	6
proba	0.48	0.28	0.24
code Huffm.	0	10	11

La longueur du code est : 12*1 + 7*2 + 6*2 = 38

3. L'entropie du texte est : 1.5166. Le codage de 25 caractères requiert donc au moins 37.91 caractères. Le code effectif est donc très proche de l'optimum.

END SOLUTION

Exercice 4 Vous voyez que le texte t contient beaucoup de répétitions, il est donc pertinent de le coder à l'aide de l'algorithme LZW. Ici, on limite la taille du dictionnaire à 8 entrées. Ceci veut dire que dans l'algorithme LZW (rappelé en annexe A.3), on ne rajoute pas de nouvelle entrée au dictionnaire si la taille maximale est atteinte, à l'étape (2(b)ii) de l'algorithme. Du reste, l'algorithme se déroule comme d'habitude.

1. Exécutez l'algorithme de compression sur le texte t. Indiquez le dictionnaire $d(t)$ qui est construit, et le code $c(t)$ obtenu.
2. Puisque le dictionnaire est limité à 8 entrées, le code du texte t est une séquence sur l'alphabet $C = \{0 \dots 7\}$. Peut-être, quelques chiffres de C n'apparaissent pas dans le code $c(t)$, mais ce n'est pas important pour l'instant. Si vous représentez les chiffres de C en binaire (avec 3 bits), le code $c(t)$ est représentable avec combien de bits ?
3. Vous constatez que les chiffres de C n'ont pas la même fréquence dans $c(t)$. Vous pouvez donc appliquer l'algorithme de Huffman à $c(t)$, à savoir : traiter les chiffres de C comme les caractères de n'importe quel alphabet, construire l'arbre de Huffman, et attribuer des codes binaires aux éléments de C selon leur fréquence dans $c(t)$. Effectuez cette démarche, et appliquez le codage de Huffman à $c(t)$. Le résultat sera un code binaire $h(c(t))$. Quel est le nombre de bits de $h(c(t))$?
4. Commentez le fait que le codage que vous obtenez est plus court que le codage optimal calculé à partir de l'entropie, voir exercice 3. N'est-ce pas en contradiction avec les théorèmes de Shannon ?

BEGIN SOLUTION

1. Le dictionnaire construit est : `{'a': 0, 'c': 2, 'b': 1, 'ba': 4, 'ca': 6, 'abc': 5, 'ab': 3, 'abca': 7}` et le texte compressé est : `[0, 1, 3, 2, 5, 0, 7, 7, 7, 7]`
2. Chaque chiffre est représenté par 3 bits, et pour le code de 10 chiffres, on a besoin de 30 bits.
3. La distribution de fréquence des chiffres et comme suit :

Chiffre	0	1	2	3	4	5	6	7
# occur.	2	1	1	1	0	1	0	4
code Huffm.	01	0000	0001	0010		0011		1

La longueur du code $h(c(t))$ est donc 6*4=24 bits.

4. Il n'y a pas de contradictions avec les théorèmes de Shannon, qui ont pour objet un codage optimal pour une source d'information qui émet des caractères d'un alphabet de manière indépendante ("sans mémoire"). Cette situation n'est pas donnée ici, où on a regroupé plusieurs caractères en séquences qui forment les entrées du dictionnaire de LZW.

END SOLUTION

3 Codes détecteurs d'erreur : CRC

Exercice 5 Soit $G(X) = X^3 + X^2 + 1$ un polynôme générateur pour un code CRC.

Utilisez ce polynôme pour

1. coder les messages [110100111]
2. décoder le message [001110110110]
3. décoder le message [111000111111]

"Décoder" un message veut dire : déterminer s'il a été transmis correctement, et extraire le message en cas de transmission correcte.

BEGIN SOLUTION

1. Pour le codage : Le message $M(X)$ est [110100111]. On calcule le reste $R(X) = M(X) * X^3 \mod G(X) = [110100111000] \mod [1101] = [1]$ Le message envoyé est donc $Env(X) = M(X) * X^3 + R(X) = [110100111000] + [1] = [110100111001]$
2. Décoder $Rec(X) = [001110110110]$ en calculant $Rec(X) \mod G(X) = [101]$. Il y a donc erreur de transmission
3. Décoder $Rec(X) = [111000111111]$ en calculant $Rec(X) \mod G(X) = [0]$. La transmission était correcte, le message est [111000111].

END SOLUTION

Exercice 6 On peut se poser la question si certains polynômes générateurs du CRC sont uniformément "meilleurs" que d'autres, c'est à dire, peuvent détecter des erreurs que d'autres polynômes ne peuvent pas détecter. Ici, un polynôme $G(X)$ détecte une erreur entre un message envoyé $Env(X)$ et reçu $Rec(X)$ si $Err(X) \mod G(X) \neq 0$, où Err est le polynôme d'erreur $Err(X) = Rec(X) - Env(X)$.

1. Soient $G_1(X) = X^3 + X^2 + 1$ et $G_2(X) = X^3 + X + 1$ deux polynômes générateurs. Indiquez un polynôme d'erreur $Err_1(X)$ que G_1 ne peut pas reconnaître, mais G_2, et inversement, un polynôme $Err_2(X)$ que G_2 ne peut pas reconnaître, mais G_1. Justifiez votre réponse. – Aucun de G_1 et G_2 est donc uniformément meilleur que l'autre.
2. Plus en général, soient $G(X)$ et $G'(X)$ des générateurs arbitraires. Montrez que le générateur produit $GP(X) = G(X) * G'(X)$ détecte toutes les erreurs détectées par $G(X)$ et par $G'(X)$.
3. Montrez que $GP(X)$ détecte même strictement plus d'erreurs et est donc uniformément meilleur que chacun de $G(X)$ et $G'(X)$, pourvu qu'aucun des deux ne soit un polynôme constant.

BEGIN SOLUTION

1. Il suffit de choisir $Err_1 = G_1$ et $Err_2 = G_2$ et remarquer que dans ce cas, $Err_1 \mod G_1 = 0$ et pareil $Err_2 \mod G_2 = 0$. Par contre, $Err_1(X) \mod G_2(X) = Err_2(X) \mod G_1(X) = X^2 + X$
2. Si $Err(X) \mod (G(X) * G'(X)) = 0$, alors $(G(X) * G'(X))$ est un facteur de $Err(X)$, donc aussi $G(X)$ est un facteur de $Err(X)$, et par conséquent $Err(X) \mod G(X) = 0$ et $Err(X) \mod G'(X)$. Par contraposée, si $Err(X)$ est détectée par $G(X)$, alors $Err(X)$ est aussi détecté par $(G(X) * G'(X))$.
3. Nous formons le polynôme d'erreur $Err(X) = G(X) * (G'(X) + 1)$. Il n'est pas une constante parce que aucun des facteurs n'est une constante. D'un côté, $Err(X) \mod G(X) = 0$ et n'est donc pas détecté par $G(X)$. De l'autre côté, $Err(X) \mod GP(X) = G(X) \neq 0$ est détecté par le polynôme produit.

END SOLUTION

A Définitions et énoncés du cours

A.1 Inégalité de Kraft

Il existe un code préfixe binaire avec k codes $u_1 \dots u_k$ si et seulement si

$$\sum_{i=1}^{k} 2^{-|u_i|} \leq 1$$

où $|u_i|$ est la longueur du code u_i.

A.2 Théorie de l'information

Entropie (selon Shannon) d'une source d'information avec des événements $\{x_i | i \in I\}$:

$$H =_{def} \sum_{i \in I} (-\log_2(P(x_i))) * P(x_i)$$

Longueur moyenne d'un ensemble (mot × probabilté) : $lnm(E) = \sum_{(m,p)\in E} |m| * p$

Notions de logarithme

- Définition du logarithme en base b : $\log_b(x) = y$ si et seulement si $x = b^y$.
- Calcul du logarithme binaire à l'aide du logarithme décimal : $\log_2(x) = \frac{\log_{10}(x)}{\log_{10}(2)}$

A.3 Algorithme LZW - compression

Entrée : Une chaîne de caractères `str`
Sortie : Une liste `compr` de positions dans le dictionnaire
Algorithme : construit en même temps

- la sortie `compr`
- le dictionnaire `dict`

1. Initialis. de `dict`, `compr = []`, mot partiel `m = ''` (chaîne vide)
2. Boucle : Pour tout caractère `c` de `str` :
   (a) si `m + c` est dans `dict`, étendre `m` avec `c`
   (b) si `m + c` n'est pas dans `dict` :
      i. Ajouter `dict[m]` à `compr`
      ii. Ajouter `m + c` à la dernière position de `dict`
      iii. Mettre `m=c`
3. Pour finir, rajouter `dict[m]` à `compr`

A.4 Codes CRC

Codage pour envoyer un message $M(X)$:

- Calculer : $R(X) = (M(X) * X^n) \mod G(X)$
- Le message envoyé : $Env(X) = M(X) * X^n + R(X)$

Décodage pour détecter une erreur de transmission

- Soit $Rec(X)$ le message reçu.
- Si $Rec(X) \mod G(X) = 0$, probablement $Rec(X) = Env(X)$
  Récupérer $M(X) = Rec(X) \div X^n$
- Si $Rec(X) \mod G(X) \neq 0$, il y a certainement une erreur