INF4710 : Introduction aux technologies multimédia

Solutions pour Exercices de préparation pour l'examen intra

1. 3A2B2A2B5A2B2G1H1F1S1W1Z3A1Z1F1S1E1F7C1A

   Le nombre de répétitions maximal est 7, donc peut être représenté sur 3 bits.
   Originalement : 39*8=312 bits
   Après RLE : 20 caractères : 20*8 + 20*3 =220 bits
   Taux de compression : 1-220/312=0.29

2. (2,4),2,2,2,2,2,2,2,4,6,6,6,6,6,6,6
   (2,13), 2,4,6

3. Séquence 1 : $\hat{x}_n = x_{n-1}$, les valeurs sont à peu près constantes
   Séquence 2 : $\hat{x}_n = 2x_{n-1}$ les valeurs doubles.

4. 10, 2,8,11,-19,0,31,0,-23,0,-10,-1,6,0 : différence max : 31, 5 bits + signe=6 bits.
   10,-3,2,1,-34.5,-6,25,-21.5,-44.5,-10,-20,-6,1.5,-7.5 : max : 44, 6 bits (si on arrondit)+ signe=7bits.
   Le premier prédicteur est meilleur. Taux de compression 1-6/8 vs 1-7/8.

5. A : 0, B : 1000, C : 101, D : 1001, E : 11 (Avec programme MATLAB)
   Message codé : 0 1000 101 0 0 1001 11 11 0 0 1000 101 0 11 11 0 0
   Taux de compression : 1-34/(17*3)=0.33

   Ou -> A : 0, B : 110, C : 1111, D : 1110, E : 10
   Voici l'arbre : http://huffman.ooz.ie/?text=ABCAADEEAABCAEEAA

6. Huffman :

   Voici l'arbre:
   http://huffman.ooz.ie/?text=ABABCAACDEEEBDCCDBE
   On 3, 4, 4, 4, 4
   On groupe 4 et 3, ça donne 7
   On a 4 4 4 7
   On groupe 4 et 4 (Les plus petits)
   On a 4 7 8
   On groupe 4 et 7
   On a 8 , 11
   On groupe 8 et 11 -> 19

   A : 100, B : 01, C : 00, D : 101, E : 11
   Message codé : 100 01 100 01 00 100 100 00 101 11 11 11 01 101 00 00 101 01 11
   Taux de compression : 1-45/(19*3)=0.21

   Code binaire tronqué :
   5=2^k+b, donc k=2, b=1, 3 symboles 2 bits, 2 symboles 3 bits
   A : 00, B : 01, C : 10, D :110, E :111

Message codé : 00 01 00 01 10 00 00 10 110 111 111 111 01 110 10 10 110 01 111
Taux de compression : 1-45/(19*3)=0.21

Les fréquences des symboles étant à peu près les mêmes, Huffman ne permet pas d'obtenir mieux qu'un code binaire tronqué, car la longueur des codes sera à peu près similaire.

7. 0.2549
   1) 0 A 0.4700 B 0.5900 C 0.7100 D 0.7700 E 1.0000
   2) 0 A 0.2209 B 0.2773 C 0.3337 D 0.3619 E 0.4700
   3) 0.2209 A 0.2474 B 0.2542 C 0.2609 D 0.2643 E 0.2773
   4) 0.2542 A 0.2574 B 0.2582 C 0.2590 D 0.2594 E 0.2609
   5) 0.2542 A 0.2557 B 0.2561 C 0.2564 D 0.2566 E 0.2574

   Dans le pire des cas, on aurait tous des D de fréquence 0.06. Donc, pour 17 caractères, on a 0.06^17, ce qui est équivalent à environ 2^-70, comme dimension d'intervalle final. Si on doit représenter un nombre qui commence par 2^-1, un nombre double précision n'est pas suffisamment précis, car il y a seulement 52 bits pour la mantisse dans le format IEEE754. On peut quand même constater que 1-70/(17*8), donne malgré tout un taux de compression de 0.485

8. AGFDIGAI
   AA -> F
   BC -> G
   EE -> H
   HF -> I
   On passe de 17 à 8 caractères, donc environ une compression d'environ 1-8/17, si le codage demeure sur 8 bits, et si on néglige la table des correspondances.

9. 000 000 001 000 0010 0100 0011 0100 0100 0000 0011 1110 00000 01010 01110 00001 00001 00010 10000 00001 01101 10110 00001

   A 00000
   B 00001
   C 00010
   D 00011
   E 00100
   AA 00101
   AB 00110
   BA 00111
   AC 01000
   CE 01001
   ED 01010
   DE 01011
   EE 01100
   EA 01101
   AD 01110
   DA 01111
   ADA 10000
   AE 10001
   EDA 10010

ADB 10011
BB 10100
BC 10101
CA 10110
ADAB 10111
BE 11000
EAC 11001
CAB 11010

Taux de compression 1-(99/30*3) = -0.1, message trop court. On n'exploite pas bien les répétitions. Donc, méthode par dictionnaire = messages longs.

10. Pour les 6 pixels du milieu (Niveaux: 32 96 160 224)

34.0000	32.0000	206.6875	67.0000	54.0000
38.4375	71.0625	202.8125	65.0000	44.0000
32.0000	72.0000	180.0000	70.0000	41.0000

34.0000	32.0000	224.0000	59.4258	54.0000
38.4375	67.8164	197.4023	63.9180	44.0000
32.0000	72.0000	180.0000	70.0000	41.0000

34.0000	32.0000	224.0000	32.0000	65.9988
38.4375	67.8164	202.5447	72.4885	45.7141
32.0000	72.0000	180.0000	70.0000	41.0000

34.0000	32.0000	224.0000	32.0000	65.9988
38.4375	96.0000	190.2144	72.4885	45.7141
26.7156	63.1926	178.2385	70.0000	41.0000

34.0000	32.0000	224.0000	32.0000	65.9988
38.4375	96.0000	160.0000	85.7073	45.7141
26.7156	68.8578	187.6805	71.8884	41.0000

34.0000	32.0000	224.0000	32.0000	65.9988
38.4375	96.0000	160.0000	96.0000	41.2111
26.7156	68.8578	185.7506	68.6719	40.3567

11.

-7	-3	-2	0
1	-3	1	0
1	2	0	0
1	0	1	0

Zigzag : -7, -3, 1, 1, -3, -2, 0, 1, 2, 1,0,0,0,0,1,0
RLE et Huffman : -7, (0,x) -3, (0,x) 1, (0,x) 1, (0,x) -3, (0,x) -2, (1,x) 1, (0,x) 1, (0,x) 2, (0,x) 1, (4,x) 1, (0,0). Note : x, indique la taille en nombre de bits selon une table de Huffman inconnue.

12. 56, -13, 5, -23, 53, -33, 4, 4, 21, -8, 0
    L’erreur max est de 53. 6 bits + signe, 7 bits, donc taux de compression de 1-7/8, si originalement les échantillons étaient sur 8 bits.

13. 56, 0, 0, 0, 1, 0, 1, 1, 1, 1 ,1 ,1

    56, valeur originale du signal. Le signal baisse ou augmente de 5.

    La valeur suivante est 43, 43 < 56, donc on baisse 0, et fait -5 (donc, le signal vaut 56-5=51).

    La valeur suivante est 48, 48 < 51, donc on baisse 0, et fait -5 (donc, le signal vaut 51-5=46)

    La valeur suivante est 25, 25 < 46, donc on baisse 0, et fait -5 (donc, le signal vaut 46-5=41)

    La valeur suivante est 78, 78 > 41, donc on augmente 1, et fait +5 (donc, le signal vaut 41+5=46)

    La valeur suivante est 45, 45 < 46, donc on baisse 0, et fait -5 (donc, le signal vaut 46-5=41)

    La valeur suivante est 49, 49 > 41, donc on augmente 1, et fait +5 (donc, le signal vaut 41+5=46)

    etc.

    Taux de compression : fixe, c’est toujours 1-(1 bits/8bits) : 0.875

14. DPCM :
    123, 6, 6, -123, 2, 2, 2, 2, 12, 10, 3, 44, -18

    Ensuite, on doit coder 6, 6, -123, 2, 2, 2, 2, 12, 10, 3, 44, -18 par Golomb-Rice.
    Il y a des nombres négatifs. Dans ce cas, avant de coder, on fait 2*x pour les nombres positifs et 2*abs(x)-1 pour les nombres négatifs. Donc, on code :

    12 12 245 4 4 4 4 24 20 6 88 35

    M=12
    Code binaire tronqué :
    0 :000 1 : 001 2 :010 3 :011 4 :1000 5 :1001 6 :1010 7 :1011 8 :1100 9 :1101 10 :1110 11 :1111

    Golomb-Rice :
    '10000' '10000' '1111111111111111111101001' '01000' '01000' '01000' '01000' '110000' '101100' '01010' '111111101000' '1101111'

15. L’audio sera en avance de 0.048 seconde.