Examen 1999/2000 Compression

Le 5/1/2000, Tout document autorisé

Jeu de dames

Soit un jeu de dames **simplifié** dont les règles sont :

- Le plateau de jeu est un pavage 8 par 8, on joue dans les cases.
- La position initiale est représentée par la figure 1.
- Blanc commence.
- Chaque joueur joue à son tour en déplaçant un pion d'une case en diagonale (figure 2, 3, 5, 6) en direction du camp adverse (pas de retour en arrière) ou en sautant par dessus une pion de l'adversaire et en le prenant (en avant ou en arrière) autant de fois qu'il le désire (figure 4, 7)
- Si un joueur ne peut pas jouer, il a perdu.
- Il n'y a pas de reine et on ne souffle pas.

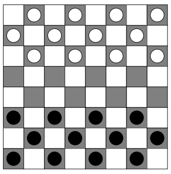
Fig 1

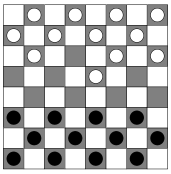
Fig 2

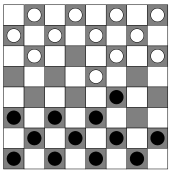
Fig 3

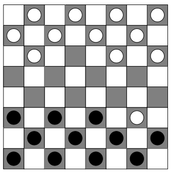
Fig 4

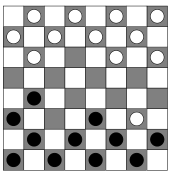
Fig 5

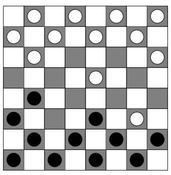
Fig 6

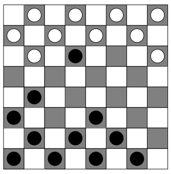
Fig 7

Toute les questions se rapportent au codage de la position du jeu à un instant t et non d'une partie complète.

Codage

Les questions 1 à 6 sont des questions de codage, il n'y a donc pas lieu de parler de HUFFMAN, LZW, ...

Les codages des questions 1 à 3 doivent occuper une place mémoire indépendante de la position. Donner la formule permettant de calculer le nombre de bits occupé par le codage et son évaluation si possible.

1. Le plateau et les pions ont été utilisés par des gens qui ne jouent pas aux dames mais qui posent néanmoins les pions noirs et blancs dans les cases (sans les empiler). Proposez la méthode de codage **la plus simple** possible.
2. Améliorez (en espace) le codage de la méthode précédente au détriment du temps de calcul.
3. Finalement, les gens jouent avec les règles indiquées. Proposez une amélioration **triviale** de la méthode précédente.

Les codages des questions 4 à 6 occupent une place mémoire dépendante de la position. On suppose que les joueurs respectent les règles.

4. Proposez un codage (le plus compacte) de la position, occupant une place approximativement linéaire en fonction du nombre de pions sur le plateau de jeu. À partir de combien de pions cette méthode est moins intéressante que celle de la question 3 ?
5. Proposez un codage de la position occupant une place qui est d'autant plus courte que la position est proche du début de la partie.
6. Comment profiter du meilleur des méthodes des questions 3, 4 et 5 ?

Égalisation des probabilités

Une seule question, mais qui compte pour beaucoup de points.

7. Appliquez ce que vous savez sur l'égalisation des probabilités à **toutes** les questions précédentes. *Limitez-vous aux méthodes simples.* N'oubliez pas de donner des exemples de codage.

Compression

8. En repartant du codage défini à la question 1. Comment compresser ce codage en utilisant une méthode de compression d'image.
9. La question 3 permet le codage de positions impossibles à obtenir en jouant en respectant les règles. Ce codage n'est donc pas optimal. Proposez un codage optimal (même si son implémentation informatique est impossible à l'heure actuelle).
10. Pourquoi la réponse à la question 5 n'est pas la réponse à la question 9 ? Essayez de trouver LA raison.

Divers

11. Donnez au moins 2 raisons pour lesquelles on ne travaille pas avec des paquets de 200000 échantillons quand on comprime/décomprime des sons en utilisant la DCT.
12. Soit la même image N&B (1 bit par pixel) stockée dans un fichier PBM (8 pixels par octet) et PGM (1 pixel par octet, donc 2 valeurs possible pour l'octet). On comprime les deux fichiers avec un algorithme fait pour compresser des fichiers texte. Que pensez-vous qu'il va ce passer ?
13. Donnez plusieurs raisons à la dégradation du taux de compression de la méthode MPEG sur des films vidéo pris dans une boite de nuit.
14. Cette feuille que vous tenez comporte grossièrement 8 millions de pixels N&B (1 million d'octets). Comment la réduire à 2432 octets sans changer la valeur d'un pixel ?
15. Est-ce qu'un algorithme spécifique pour comprimer du HTML serait plus performant qu'un algorithme général ? Pourquoi ? Quels en seraient les inconvénients ?
16. Citez plusieurs type de données qui ne peuvent pas être comprimées.
99. Combien de bits pour coder une date précise au jour près entre le 01/01/1900 et le 31/12/1999: en texte lisible, en BCD, avec 3 entiers de tailles différentes, avec 1 entier. D'où vient le bug de l'an 2000 ?

Erreurs courantes faites

Vous pensez tous à “comment coder” sans vous demander si le “décodage” est possible.

1 En langage C : sizeof(t[8][8]) == sizeof(t[64])

4 Si le codage est de longueur variable, sa longueur doit être codée dans le codage si cela n'est pas intrinsèquement fait.

8 Le fait de coder le nombre de valeur nulles suivie de la valeur non nulle n'est pas une méthode de compression d'image mais de facteur de DCT. Pour des images, cette méthode est très mauvaise.

La compression des FAX est sans perte.

11 Le fait que 200000 n'a pas de racine entière ne change rien du tout. Si l'on applique pas la psycho acoustique, il n'y auras pas de perte de qualité (on ne peut appliquer la psychoacoustique car l'échantillon dure plus de 0.1 seconde)

12 La compression de fichiers texte est toujours sans perte même si elle est appliquée à des fichiers contenant des images. D'autre part vous avez tous considéré le facteur de compression au lieu de considérer la taille finale du fichier.

Remarque : Il n'y a aucune raison de faire des suppositions sur la probabilité des valeurs des pixels. À la limite sur leur organisation.

Remarque : Si les événements d'un fichier sont équiprobables, cela n'empêche pas de le comprimer (par exemple 100 bits à 1 suivi de 100 bits à 0).

13 Stroboscope (pas stromboscope ou strombinoscope ou ...)

14 Si l'on fait une compression d'image classique, on arrive à 200ko en GIF et 100Ko en FAX. Qui vous a dit que je travaillais en LaTeX ?

15 Un entier seul peut être comprimé, en effet, on peut le remplacer par son code de Huffman. Mais de temps en temps, la taille de l'entier comprimé sera plus grande que la taille de l'entier initial pour les entiers de valeur improbable.

99 En BCD, on ne code que les nombres de 0 à 9 sur 4 bits, on ne code pas 10, 11, ...

Remarque

Vous n'avez réellement aucun moyen de savoir si la RAM contient en interne (physiquement) des valeurs 0,1 en voltage pour stocker un bit ou bien 0,1,2,3 pour en stocker 2 à la fois.

Correction

1 Dans un Mail vous notez :

```
-B-B-B-B
B-B-B-B-
-B-B-B-B
--------
--------
N-N-N-N-
-N-N-N-N
N-N-N-N-
```

Le codage se fait simplement en remplaçant chaque caractère par deux bit : - par 00, N par 01, B par 10. Le codage occupe donc 2*64 = 128 bits.

2 La chaine de bit 11 est inutilisée. Pour ne pas perdre de place, il suffit de considérer que la position est un nombre exprimé en base 3. On prend -=0, N=1, B=2 et on convertit l'entier de base 3 en binaire. Si C_i est le contenu de la case i. On représente la partie comme l'entier : $\sum_{i=0}^{63} 3^i C_i$ Le temps de calcul est plus long car on doit faire des multiplications et divisions sur des entiers de tailles importantes. La taille prise est de $\log_2 3^{64}$ soit 102 bits.

3 Si l'on respecte les règles, il n'y a jamais rien sur les cases noires. On code donc sur 51 bits.

4 Pour les blanc puis pour les noirs, on indique le nombre de pions (4 bits), puis pour chaque pions la position XY de la case sur laquelle il est (5 bits). La place occupée est donc de : $2 * 4 + n * 5$. Pour 9 pions cette méthode prend trop de place. Remarque : si l'on code un nombre de pions total et 1 bit pour coder la couleur de chaque pion, la place occupée est de $5 + n * 6$ ce qui est moins bien que la méthode précédente.

5 Deux solutions. Je ne sais pas laquelle est la meilleure.

a Vous codez le nombre de coups joués puis pour chaque coup son index dans la liste des coups possibles dans la position. Grossièrement, si dans la partie il n'y a pas plus de 32 coups possibles à un moment donné on peut utiliser 5 bits par coups. (En fait, à un moment donné il peut y avoir des centaines de coups possible mais c'est vraiment tordu comme configuration) Avec cette méthode, on ne code que des positions possibles.

b Vous codez les changements par rapport à la position initiales. Pour cela, vous coder le nombre de changements (5 bits) puis pour chaque changement, sa position (5 bits) et la nouvelle valeur (1 bits). Il suffit d'1 bit pour coder la nouvelle valeur car il n'y a pas 3 choix possibles (sinon il n'y aurait pas de changement).

6 Vous codez la position avec les 3 méthodes, vous prenez la meilleur et en début de codage vous indiquez la méthode employée.

7 1 On affecte les chaines de bit 0, 10 et 11 aux trois états possible en fonction de leur nombre d'occurrences. On code l'arbre dans le codage.

2 On ne travaille plus en base 3 mais on fait un codage arithmétique. En fait, en travaillant en base 3 on fait un codage arithmétique de 3 événements équiprobables.

3 Idem.

4 Les coordonnées verticales des pions ne sont pas équiprobables. On peut donc gagner de la place, surtout en début de partie.

5a On code avec des entiers de longueur variable, et on suppose que les joueurs jouent bien, ont trie donc les index du "meilleur coup" (0) ou plus mauvais.

6 On essaye de voir si le codage utilisé le plus courrament afin de l'indiquer par 1 seul bit.

8 Par exemple en faisant du RLE. On code le nombre de séries, et pour chaque série : le nombre de cases (3 ou 4 bits) et son contenu (1 ou 2 bits)

9 On parcourt toutes les parties possibles en les indexant sans indexer 2 fois la même position. Ce codage est optimal bien qu'il puisse prendre plus de place que le codage de 6) ou 8) dans certain cas. En fait, on peut l'améliorer en donnant les numéros les plus petits au parties les plus probables (les mieux jouées)

10 Parce que deux codages différents de la question 5 peuvent coder la même position.

11 Le temps de calcul est en n^2, le temps de latence est trop important avant de transmettre le son (10 secondes), on ne peut appliquer la psychoacoustique car le son analysé dure plus d'1/10 de seconde.

12 Le fichier PGM ne contient que 2 événements avec beaucoup de répétitions, il sera donc fortement comprimé. Le fichier PBM contient 256 événements, les répétitions seront plus rare (cela dépend du fichier), il sera donc pas très bien comprimé. Mais comme le fichier PBM est au départ 8 fois plus petit : *on ne peut rien conclure.* J'ai testé sur un exemple, le fichier PGM.gz était 2 fois plus gros que le PBM.gz.

13 Les stroboscopes font échouer la réutilisation de l'image précédente et les éclairages colorés diminuent l'efficacité du JPEG car le passage en YUV est fait pour de la lumière blanche.

14 Pour comprimer, on essaye de retrouver le modèle, or, pour cet énoncé le modèle (document source) existe. Il suffit de comprimer le source du document pour obtenir la taille indiquée. Concernant la valeur numérique, elle est fausse car elle s'appliquait à un énoncé où les diagrammes étaient dessinés avec des B, N, -. Mais cela ne change pas grand chose.

15 Beaucoup de raisons permettent de mieux comprimer : un document HTML est un arbre (codage spécifique), les *tags* ne peuvent pas être mis n'importe où (dans un bloc UL on met des LI et pas d'autres *tag*), dans les tags les blancs et la capitalisation ne compte pas, les codes caractères sont connus, le nombre de blancs et de retour à la ligne ne compte pas la plupart du temps, ... Avec un tel compresseur, du HTML présenté n'importe comment serait décompressé en du HTML joliment présenté (indentation, ...). Inconvénients : le source du document est modifié mais pas sa sémantique, Remarques : le compresseur peut tenir compte de *tags* qu'il ne connait pas et comprimer des zones invalides de HTML comme du texte normal.

16 Les fichiers déjà comprimés, les fichiers contenant des valeurs aléatoires équiprobables.

19 Remarquez bien que la question demande un date entre le 01/01/1900 et le 31/12/1999 et non entre le 01/01/00 et le 31/12/99. Donc il n'y a pas de problème de saisie de la date (ce qui élimine la grande majorité des bugs de l'an 2000). Par contre, je demande le codage entre ces 2 dates, il n'est donc pas nécessaire de coder les deux premiers chiffres.

- En texte lisible : 6 caractère (JJMMAA) = 48 bits
- En BCD : 6 codes BCD (JJMMAA) = 24 bits
- Avec 3 entiers : Jour (5 bits), Mois (4 bits), Année (7 bits) = 16 bits.
- Avec 1 entiers : 100*365.25 = 36525 = 16 bits.

Supposons que le bug de l'an 2000 ait été du à un besoin d'économiser de la place. Les deux derniers codages auraient été utilisés. Bizarrement, le codage avec 3 entiers peu coder les dates jusqu'au 31/12/2027 et celui avec un entier peu les coder jusqu'en 2079. Donc ce n'est pas a cause de la place.

Le bug a lieu en l'an 2000 car ce sont les deux premier codages qui ont été utilisés. Ils l'ont été car les langages de programmations sont orientés caractère (ou chiffre) et qu'utiliser un autre codage compliquait la tache du programmeur en le forçant à faire des convertions supplémentaires.