Examen 2000/2001 Compression

Le 19/12/2000, Tout document autorisé

Compression HTML

- Quels intérêts à comprimer du HTML ?
- Comprime-t-on de la même manière pour l'archivage ou pour la transmission ?
- Si l'on désire utiliser un compresseur de texte classique, quels prétraitements simples doit-on faire pour augmenter le taux de compression ? Répondre dans le cas de l'archivage et dans celui de la transmission.
- On s'autorise à ajouter des informations autres que du HTML dans le fichier ou à utiliser de fichiers séparés. Proposez d'autre méthodes de prétraitement pour gagner de la place dans le cas de fichiers HTML complexes.
- Malgré tous ces prétraitements, pourquoi le compresseur de texte ne va pas comprimer de manière optimale le fichier HTML ?
- On suppose que le fichier HTML vérifie parfaitement la norme. Proposez une méthode de codage compacte du fichier HTML (codage, pas compression).
- On veux comprimer le codage précédent, détaillez précisemment tout ce qu'il convient de faire (question importante).
- Améliorer la méthode précédente dans le cas de pages multilingues (avec l'attributs `LANG` sur les tags).
- La méthode de compression que vous avez proposée est certainement très bien, mais comme personne n'a le décompresseur cela ne sert pas à grand chose... Comment détourner ce problème ?

Compression d'image du ciel de nuit

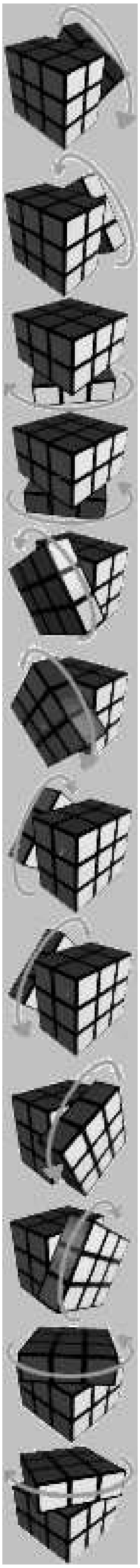
Un astronome désire archiver des images du ciel dans les conditions suivantes :

- Objectif grand angle permettant de prendre une grande partie du ciel
- Résolution des images pouvant être très importante.
- Ciel sans nuage.
- Au coeur de la nuit.
- On prend une image par minute.
- Compression sans perte d'information.

Questions :

- Quel différence entre une image de ce type et une image classique ?
- En quoi ces différences jouent sur la compression JPEG ?
- Quel facteur de qualité doit-on prendre pour la compression JPEG ?
- Quel prétraitements doit-on faire avant de faire la compression JPEG pour augmenter **fortement** le taux de compression ? On suppose que pour les questions suivantes on utilise ce prétraitement.
- Y-a-t-il un différence entre la méthode de compression de la première image et des suivantes ? (Si oui laquelle, si non pourquoi)
- Avez-vous une idée de la taille d'une image comprimée ?
- Certaines images comprimées sont plus grosses que d'autres, dans quels cas ?
- Les images sont prises à partir d'un satellite, cela change-t-il quelque chose ?

Rubiks cube



Le cube classique est de taille 3x3x3. Extérieurement on voit 8 petits cubes de coins, 12 petits cubes d'arête et 6 petit cubes de centre de face. Le cube central n'est pas visible. Sur chaque face externe d'un petit cube est collé une gommette d'une couleur parmi 6 possibles. Lorsque vous achetez le cube, les 6 faces sont de couleurs unis différentes, dans la suite de l'énoncé, on l'appellera un "vrai cube". Chacune des 6 faces du cube peut subir une rotation d'autant de quart de tour que l'on désire autour de son centre. Après quelques rotations, les faces du cube sont généralement coloriées "au hasard" si l'on avait un "vrai cube" au départ, le résultat est toujours un "vrai cube" (un "vrai cube" est un cube que l'on peut refaire). On s'intéresse au codage de l'état du cube "mélangé", on ne s'intéresse pas à sa position dans l'espace (face blanche vers moi par exemple).

- Combien y-a-t-il de gommettes sur un cube ?
- Un employé fou de l'usine colle les gommettes en choisissant les couleurs au hasard. Proposez un codage simple et un compact qui permette de coder les cubes qu'il fabrique.
- Les gommettes de votre cube se sont décollées, vous les recollez au hasard. Proposez un codage simple et un compact qui permette de coder votre cube.
- A partir du cube de la question précédente, pouvez-vous dans tous les cas, en manipulant le cube, donner des couleurs uniformes aux faces ? (Donnez un contre exemple).
- Votre cube s'est cassé en morceaux, tous les petits cubes sont tombés par terre sauf celui du centre et les 6 centres de face qui forment un bloc et dont les positions respectives ne peuvent pas changer. Vous le remontez sans faire attention aux couleurs. Proposez un codage simple et un compact qui permette de coder votre cube.
- On veut maintenant coder un "vrai cube". Proposez un codage simple (mais pas forcément facile à calculer) qui ne permette de coder QUE des "vrais cubes".
- Le codage précédent est-il minimal en place mémoire ?
- Quel est le codage théorique optimal d'un "vrai cube" ?
- Donnez un majorant de la taille de ce codage optimal.

Et maintenant une correction.

Compression HTML

1) Vitesse de transfert et gain de place sur les disques du serveur.

2) Pour la transmission on veut que le résultat visuel soit identique, pour l'archivage on veut que le source soit identique.

Petit points moins importants dans le cas de la transmission :

La décompression ne doit pas être trop lente

Il serai bien de décomprimer en premier les URL afin de pouvoir précharger toutes les inline avant la fin du transfert du texte.

3) Pour l'archivage enlever les blancs inutiles, normaliser tous les tags et les noms d'attributs ainsi que certaines valeurs (tous mettre en majuscule) mettre des tabulations ou c'est possible. Pour la transmission on peut en plus enlever les commentaires, l'indentation, les retours à la ligne inutiles.

4) On peut détecter les régularités dans la page HTML et les remplacer automatiquement par un CSS ou du JavaScript.

5) Par ce que le compresseur de texte ne sait pas qu'il comprime un arbre de TAG et qu'il ne fait pas la différence entre TAG et texte.

6) On code l'arbre HTML les noeuds sont des TAGS codés par un numéro. On a besoin de coder un arbre n-aire de tags

- un numéro de tag
- 1 bit pour indiquer si l'on a un frère suivant suivi du codage du frère.
- 1 bit pour indiquer si l'on a un fils suivi du codage du premier fils.
- 1 bit pour indiquer si l'on a un attribut suivi du codage du premier attribut.
- Si le tag peut contenir du texte, on code la chaine de caractère.

Pour les attributs :

- un numero d'attribut
- une valeur d'attribut
- 1 bit pour indiquer si l'on a un attribut suivant suivi du codage de l'attribut.

Pour plus d'info, voir les documents concernant le WML binaire (WML est la version XML pour WAP).

7) On utilise un Huffman statique pour coder les numéros de tag et d'attribut de manière contextuelle. En effet, dans le cas du HTML 4.0 en supposant les TAG et attributs équiprobables. Si l'on est dans un TAG <TABLE> il ne faut que 3 bits pour coder les éléments à l'intérieur de celui-ci (8 possibles) et 5 bits pour coder les numéros d'attributs (il y en a 24). Pour les valeurs d'attributs, certaines sont en nombres fixes, des entiers, pourcentages, quelconque (dans ce cas on fait de la compression de texte standard)

Si de plus on fait une analyse statistique, on pourra trouver un meilleur arbre de Huffman.

Pour le texte, on utilise un compresseur de texte standard.

8) On séparera les instances du compresseur de texte standard suivant la langue pour optimiser les arbres. On doit aussi le faire pour les attributs textuels.

9) On écrit le décompresseur en JavaScript. Comme cela la décompression sera transparente pour l'utilisateur.

Compression d'image du ciel de nuit

1) Le ciel de nuit n'est pas du bruit : du bruit, c'est par exemple la neige sur l'écran de la télé sur un canal inoccupé.

Une image classique consiste en des pages de couleurs et des dégradés ainsi que des lignes contrastés. L'image du ciel est une surface quasiment noir (sauf voie lactée) avec des points de forts contraste dont on ne veut pas perdre l'information luminosité.

2) JPEG fonctionnera normalement mais l'endroit ou il perdra le plus d'informations sera l'endroit qui nous intéresse : les étoiles.

3) Facteur qualité : 0 pour ne rien perdre (1 bit ou 2)

4) Prétraitement :

- Faire la différence entre l'image réel et une image de synthèse du ciel. Ces deux images doivent être **parfaitement identiques** si le modèle du ciel est correcte (les imperfections de l'objectif peuvent être codées dans l'algorithme faisant l'image de synthèse). Le seul point de différence est du aux perturbations atmosphériques.
- Si les perturbations atmosphériques sont importantes : on peut associer un léger déplacement de position aux étoiles.

5) Non car on ne code pas en relatif.

6) S'il y a peu de perturbations atmosphériques, je dirais 2 ou 3 coefficients par block de pixel (ne pas oublier que les étoiles/planètes/lunes ont déjà été enlevées et donc que l'image est presque toute noir).

7) Celles ou un avion passe :-)

8) Les images seront mieux comprimées car il n'y a pas de perturbations atmosphérique.

Rubiks cube

Remarque générale : il n'était pas nécessaire de faire les applications numériques.

1) 6*9 = 54 gommettes.

2) Codage du cube du fou : une suite de couleur pour un balayage donné. Simple donne $3 * 54 = 162$ bits (une couleur est codée sur 3 bits), compact donne $log_2 6^5 4 = 140$ bits (on code le cube comme un nombre en base 6)

3) Gommettes recollées au hasard : Pour chaque couleur on indique la position des gommettes de cette couleur. On le fait pour toutes les couleurs sauf la dernière qui va dans les emplacements restant. Simple donne $45 * 6 = 270$ (il faut 6 bits pour coder la position d'une gommette sur le cube), compact donne $log_2 54^4 5 = 258$.

Cette solution est largement moins bonne que la précédente car il y a beaucoup de positions différentes, on peut l'améliorer un peu en réduisant le nombre de position possibles au fur et à mesure. Simple donne $22 * 6 + 16 * 5 + 7 * 4 = 240$, compact donne $log_2 54! = 219$

On peut aussi l'améliorer en codant la position de collage en relatif par rapport à la position précédente, les entiers maximums manipulés sont plus petits de 9.

En fait, on peut faire mieux en utilisant une méthode complètement différente. En effet la méthode précédente donne un ordre de collage qui est inutile. Le codage peut être fait en donnant un masque de bits pour chaque couleur. Il faut donc : $54 + (54 - 9) + (54 - 2 * 9) + (54 - 3 * 9) + (54 - 4 * 9) = 180$ bits. Ce codage particulier ne peut être décliné en codage simple et compact.

Comme cela n'est pas normal de trouver moins bien (car il y a moins de cubes possibles) il faut améliorer le premier codage. En effet, au fur et à mesure que l'on utilise des gommettes il faut moins de bits pour coder la couleur. Ce codage est complexe et de plus sa taille dépend de l'arrangement des gommettes sur le cube. Dans le pire des cas (les 6 dernières gommettes à placer sont de 6 couleurs différentes) on gagne 3 bits pour la dernière, 2 bits la précedente, 1 bits pour les 2 d'avant soit en tout 8 bits :-(par rapport au codage de 2). Dans le meilleur des cas on gagne beaucoup plus.

4) Si vous collez 3 gommettes blanches sur un même coin...

5) Codage simple : un numéro de coins sur 3 bits + 2 bits d'orientation, un numéro d'arètes sur 4 bits + 1 bit d'orientation. On code une succession donné de petits cube, donc en tout $8 * (3 + 2) + 12 * (4 + 1) = 100$ bits. Codage compacte $log_2((3 * 8)^8) + 12 + log_2(12^12) = 92$ bits.

On peut faire plus compacte en remarquant que le choix diminue quand on les places. Simple : $4 * (3 + 2) + 2 * (2 + 2) + 1 * (1 + 2) + 1 * (0 + 2) + 4 * (4 + 1) + 4 * (3 + 1) + 2 * (2 + 1) + 1 * (1 + 1) + 1 * (0 + 1) = 78$ bits. Compact : $log_2((3^8 * 8!) * (2^12 * 12!)) = 69$ bits.

6) On code le cube par la liste des rotations qui permettent d'obtenir des faces de couleurs uniformes. Il y a 6*3 rotations possibles. On considère que dans tous les cas on peut refaire le cube en 25 rotations, ce qui donne dans le pire des cas avec un codage compacte $log_2(18^25) = 105$ bits.

7) Il n'est pas minimal car on code une information qui n'est pas utile pour coder l'état : la manière de faire le cube.

8) Codage optimal : On numérote tous les vrais cubes possibles et on code le cube par son numéro d'ordre. Le nombre de cubes différents étant de 43252003274489856000 il faut 66 bits pour le coder. Ceci veux dire que si vous remontez le cube cassé au hasard vous avez un chance sur 12 que cela soit un vrai cube. *Je ne sais pas comment on calcule le nombre de cubes différents.*

9) Le majorant de la taille optimale (66) est le plus petit codage que vous pouviez trouver : 69 dans mon cas.