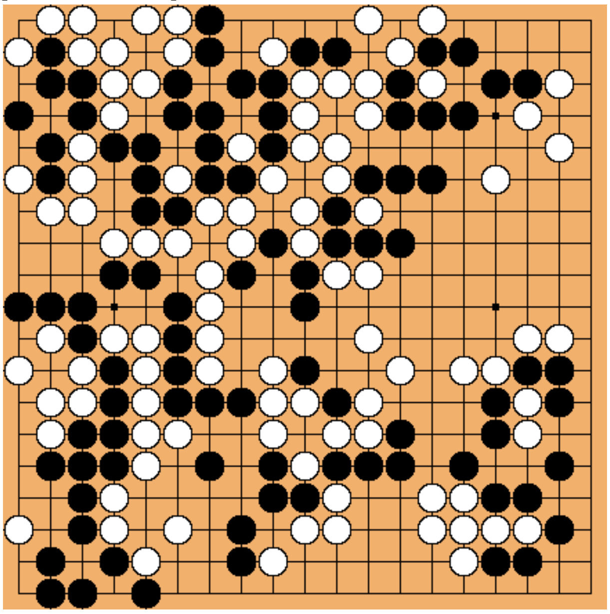
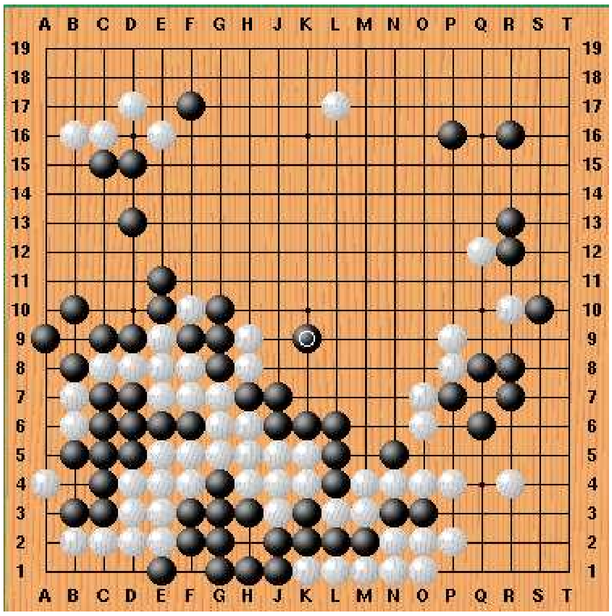
Examen 2001/2002 Compression

Le 08/01/2002, Tout document autorisé

Problème

On désire permettre aux gens de jouer au go **contre un programme** sur un site Web. Voici à quoi ressemble une partie de go, dans deux graphisme différents :





On offre à l'utilisateur les possibilités suivantes :

- Il peut cliquer sur l'endroit ou il veut jouer.
- Il peut cliquer sur «Reprendre son coups», «Abandonner», ...

On s'intéresse à une solution réalisable et portable. On utilise que du HTML pur (ni JavaScript, ni *Cookies*, ni CSS, ...) généré par un script CGI. Quand le mot «taille» est employé, il désigne la place occupée en octets et non une résolution d'image ou une taille en centimètres.

1. Détaillez la méthode **la plus simple** que vous pouvez employer.
2. Évaluer très grossièrement la taille de l'image pour les deux graphismes.
3. Évaluer la taille des données à envoyer au serveur quand on clique.
4. Quel astuce faut-il employer pour réduire la taille occupée par l'image ? (Cette astuce permet aussi d'utiliser les caches Web de manière efficace).
5. Évaluer la nouvelle taille de la page complète.
6. Peut-on diminuer la taille du transfert vers le serveur ?
7. Si oui, évaluer la nouvelle taille.
8. Quelle information est redondante ?
9. Si finalement on s'autorise à utiliser JavaScript, comment s'y prendra-t-on ?
10. Évaluer la taille finale de la page.

Questions

Analyser les problèmes suivants qui vous demandent de comprimer au mieux :

1. Les nombres (de 0 à 36 inclus) qui sont sortie d'une roulette de casino (sans tricheurs).
2. Les températures prises minute par minute sur votre balcon.
3. Une hiérarchie de fichier.
4. Un morceau de musique joué au piano.
5. Un dessin animé.

Correction

Problème

1. Le serveur ne connais pas l'état de la partie, la partie est donc stockée dans l'URL.
   Le script CGI génère une page Web contenant :
   - un tag image dont l'URL contient toutes les informations pour générer l'image avec un script CGI.
   - un formulaire dont l'action contient l'état courant de la partie.
   - une "client image map" dans le formulaire (lourd) ou bien une "server image map" qui envoi seulement la position du pixel cliqué dans l'image.
   - Les boutons de l'interface utilisateur qui enverront automatiquement au serveur (à cause de l'URL action) l'état courant de la partie.
2. Ce qui intéresse est la taille de l'image transférée et non du modèle. La première contient des zones uniformes et sera facilement compressable avec des algorithmes genre RLE suivi d'un LZW (GIF). En aucun cas du JPEG qui détruit les à plat de couleurs. Dans la réalité la taille est entre 3 et 5 Ko. La deuxième ce comprimera facilement avec LZW car contrairement à une image réelle il y a beaucoup de répétitions. On peut évaluer la taille de la première en comptant le nombre de bandes colorées qui peuvent facilement être codée sur quelques bits car elles ont toujours les mêmes longueurs. Dans la réalité la taille est entre 20 et 40 Ko.
3. Quand on clique sur reprendre son coup ou pour jouer, on doit envoyer la partie au serveur pour qu'il puisse jouer. La partie peut facilement être codée comme un nombre en base 3 puis recodé avec des caractères normaux (qui ne nécessitent pas d'être traduit avec un code hexadécimal). Si l'on considère qu'il y a 2*26+10 caractères normaux le nombre de caractères à transmettre est $ln_{62}3^{3}61 = 97$. La position du coup joué est sur 2 caractères.
4. Au lieu d'envoyer un tag contenant l'image qui doit être calculée, on envoit un code HTML contenant un matrice de petites images. Les petites images étant toujours les même elles restent dans les caches. De plus le server n'a plus besoin de calculer une grande image.
5. La page complète contient maintenant une matrice de TAG. <A HREF="34"><IMG SRC="B.gif"></A> Il y a 361 de ces TAG donx la page pèse 361*34 = 12274 octets. Cette méthode n'est intéressante que si l'image du jeu prend plus de place que cela.
6. Au lieu d'envoyer la partie au serveur, on envoi un identificateur de partie. Ceci est "dangereux" car on peut donner dans l'URL (en tapant l'URL à la main) l'identificateur de partie de quelqu'un d'autre. Le serveur devra conserver la partie complète.
7. Plus l'identificateur est court, plus un risque de vol de partie est grand.
8. L'image (ou la matrice d'image) et l'URL de l'action contiennent des informations représentant l'état de la partie. L'une peut être trouvé à partir de l'autre et inversement. Les valeurs des tags sont aussi redondantes.
9. Ce qui prend de la place est la matrice de TAG. On peut automatiquement générer celle-ci à partir d'une chaine de caractère décrivant la partie.
10. Par exemple N pour Noir, B pour Blanc, V pour vide si l'on veut rester simple. La page HTML ne contient maintenant que 413 octets environ.
    ```
    <SCRIPT SRC="go.js"></SCRIPT>
    <SCRIPT>p("VVVVVBNVVVVVVV .... ")</SCRIPT>
    ```
    L'interativité sera donc beaucoup plus agréable car la page est toute petite. L'inconvénient est qu'il faut que le client interprète JavaScript rapidement.

Questions

1. Un série de nombres aléatoires ne peut être comprimée car :
   - On ne peut pas retrouver de modèle car il n'y en a pas.
   - Tout fichier serait comprimable hors c'est impossible.
   - On pourrais comprimer un fichier comprimé car après le Huffmann on a une série "aléatoire" de 0 et de 1 si les événements sont indépendants.
   - Si vous prenez des séquences courtes (1000) pour que les nombres d'occurrence ne soient pas identiques vous gagnez de la place en codage des données mais perdez de la place pour coder l'arbre.

   La compression se résume au codage. On peut faire un codage Huffman statique (codes de 5 ou 6 bits) ou bien utiliser le codage arithmétique qui fera un peu mieux mais est plus lourd.
   Remarque : le codage de la différence fait gagner de la place seulement si le signal est continu ce qui n'est pas le cas pour la roulette.
2. La première question que l'on doit se poser est la précision de la prise de température.
   - Si c'est le degré il sera très improbable que la température varie de plus d'un degré. La plupart du temps la température sera identique sur plusieurs minutes ou dizaines de minutes.
   - Si c'est le dixième de degré, la température variera plus souvent, par contre la dérivé étant continue, on n'aura souvent des variations continues. Si la température augmente, alors elle aura tendance à continue à augmenter de la même manière.

- Si c'est le centième de degré, la température variera toutes les minutes, il y aura certainement un signal avec beaucoup de bruit.

Dans tous les cas, on code la nouvelle température par rapport à la précédente. On doit être capable de coder les changements de température improbables sinon on ne peut pas faire la compression Voici un codage de base pour le dixième de degré :

Chaines de bits pour de rares changements (1 ou 0.1 degré par minute) :

- 0 : Pas de changement de température
- 11 : +0.1 degré (on suppose que +0.1 est plus probable que -0.1)
- 100 : -0.1 degré
- 101 : Variation plus grande : on code une température absolue.

Améliorations :

- Si la précision est le degré, il peut être intéressant de faire du RLE pour coder les suites de 0 identiques.
- Tenir compte du fait que la température augmente ou diminue par plage (elle ne fait pas le yoyo de minutes en minutes sauf peut-être pour le centième de seconde) On codera donc 11 : la température varie dans le même sens que la dernière fois. 100 : la température varie en sens inverse.
- S'il n'y a pas trop de bruit, on peut essayer une méthode de prédiction de la température et coder la différence à la température prédite.

3. Coder l'arbre ne prend que quelques bits (voir cours). Pour les autres données, on peut utiliser un LZW différent par type de données. Pour les noms de fichiers, les droits, les dates d'accès, les types de fichiers, ... On peut aller plus loin en essayant de changer l'ordre des fichiers pour que la compression se passe au mieux. Mais attention certains évenement ne sont pas indépendant, un fichier ".c" ne sera jamais exécutable par exemple.
4. On code évidemment la façon de jouer et non le son produit. Le codage contient une suite d'événements dont les paramètres sont par exemple (simplifié) :
   - Le numéro de touche ou pédale
   - La date d'appui
   - La vitesse d'appui
   - La date de début de relachement

   Avec ceci on passe un morceau dont la reproduction ne tient pas compte du piano utilisé. Pour comprimer encore plus tout ceci, on peut employer plein de techniques différentes.
   - LZW pour les numéro de touche car il y a beaucoup de répétitions.
   - Un codage relatif pour les date d'appui et de la prédiction car normalement on n'appui pas à n'importe quel moment.
   - ...

   Il faut remarquer que ce codage est déjà très compacte par rapport à la compression du son du piano.
5. Remarques concernant les dessins animés classique (pas en images de synthèse) :
   - La compression JPEG (et donc MPEG) dégrade fortement les images contenant des zones de couleur uniforme.
   - Les dessins animés sont rarement en 25 images secondes.
   - Il y a souvent des répétitions exactes de séquences d'image.
   - Le fond est souvent statique, il peut être translaté ou zoomé.

   On peut imaginer comprimer un dessin animé en :
   - Trouvant les surfaces homogènes (ou à texture fixe) élémentaires.
   - En composant des groupes de surfaces homogènes.
   - Une image du dessin animé est une supperposition de groupe de surface avec un facteur de translation, homothétie.