Compression d’images à l’aide de Tetra-Arbres

Projet Algorithmique GMM2

Pierre Colomb
http://pierre.colomb.me

Responsable du Cours *Gaelle Loosli*

L'objectif de ce projet est d'implémenter en C++ une méthode de compression d'images à l'aide de *tétra-arbres*.

Vous pourrez trouver des images exemples à l'adresse http://pierre.colomb.me/teaching.html

Pour simplifier le problème, on considère, dans un premier temps, uniquement des images carrées de n par n pixels où n est une puissance de 2 (ie, il existe k avec $n = 2^k$).

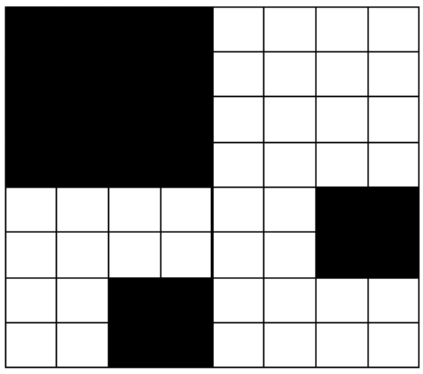
Le format PGM

On utilise des images au format PGM, qui se compose comme suit:

- Un nombre magique (P2)
- Largeur de l'image
- Hauteur de l'image
- La valeur maximale utilisée pour coder les niveaux de gris (Doit être inférieure à 65536).
- Données de l'image : L'image est codée ligne par ligne en partant du haut, Chaque ligne est codée de gauche à droite. Chaque pixel est codé par une valeur numérique. La valeur 0 correspond à un pixel noir. Un pixel blanc est codé par la valeur maximale. Chaque niveau de gris est codé par une valeur entre ces deux extrêmes, proportionnellement à son intensité.

On peut considérer par exemple l’image suivante en noire et blanc et le fichier PGM la représentant.

```
P2
8
8
1
0  0  0  0  1  1  1  1
0  0  0  0  1  1  1  1
0  0  0  0  1  1  1  1
0  0  0  0  1  1  1  1
1  1  1  1  1  1  0  0
1  1  1  1  1  1  0  0
1  1  0  0  1  1  1  1
1  1  0  0  1  1  1  1
```



Tétra arbre

Un *Tétra-arbre* est un arbre où chaque noeud différent d'une feuille possède 4 fils.

On considère par conséquent deux types de noeuds : les noeuds internes et les feuilles auquels on associe respectivement les 4 fils du noeud ou une couleur.

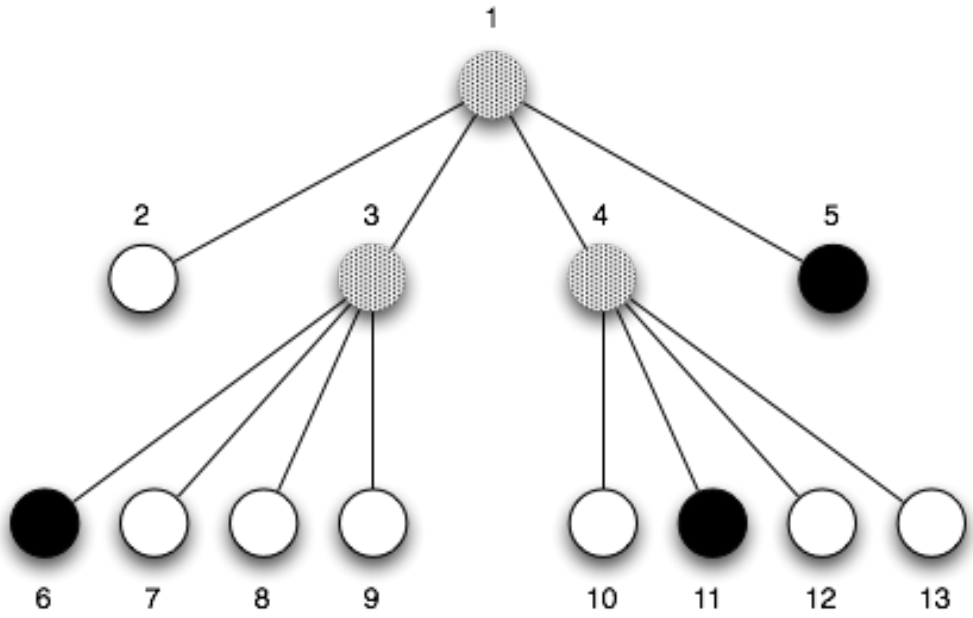
Ils peuvent être représentés dans un fichier texte à l'aide du codage suivant

- Un nombre magique (A4)
- Nombre de noeud
- La valeur maximale utilisée pour coder les niveaux de gris
- Chaque noeud est représenté par une ligne
  `x[`x_1`,`x_2`,`x_3`,`x_4`]` si x_1,x_2,x_3,x_4 sont les fils du noeud x
  `x[c]` si x est une feuille de couleur c

En supposant une image ayant au moins deux couleurs, elle peut se représenter par un tetra-arbre comme suit : la racine de l'arbre correspond à l'image complète, puis les 4 fils de la racine correspondent chacun à un quart de l'image. Par convention on considère que les 4 fils sont ordonnés dans le sens des aiguilles d'une montre à partir du quart supérieur droit de l'image. Puis, le processus est réitéré en considérant chaque sous image indépendamment.

On peut considérer la représentation de l'image précédente sous forme de tetra-arbre ainsi que la représentation du tetra-arbre sous forme d'un fichier texte.

```
A4
13
1
1[2,3,4,5]
2[1]
3[6,7,8,9]
4[10,11,12,13]
5[0]
6[0]
7[1]
8[1]
9[1]
10[1]
11[0]
12[0]
13[0]
```



Algorithme

La construction d'un *tetra-arbre* à partir d'une image *I* peut donc ce faire à l'aide de l'algorithme récursif suivant qui prend en paramètre l'image et retourne la racine du *tetra-arbre* construit.

Pour simplifier les notations on considère que pour une image *I*, les sous image I_1, I_2, I_3 et I_4 correspondent au sous images disposées dans le sens des aiguilles d'une montre à partir du quart supérieur droit de l'image.

```
Compresse(I)
  Début
    Si Uniforme(I) faire
      Retourner(Noeud(Couleur(I)))
    Sinon
      X = Compresse(I1)
      Y = Compresse(I2)
      Z = Compresse(I3)
      T = Compresse(I4)
      Retourner(Noeud(X,Y,Z,T))
    fSi
  Fin
```

Où l'on considère que la fonction `Noeud()` créé soit une feuille de couleur *c* soit un noeud interne aillant les 4 noeuds passé en paramètre comme fils. La fonction *`Couleur()`* retourne la couleur relative de l'image dépendant de la fonction `Uniforme()` utilisée.

La fonction `Uniforme()` peut être implémentée de différentes façons

- Une image est uniforme si et seulement si tous ses pixels sont de la même couleur. La couleur relative de l'image est donc l'unique couleur présente dans l'image

- Une image est uniforme si et seulement si $(c_{max} - c_{min}) / max < k$, où c_{max} et c_{min} sont les valeurs maximum et minimum des couleurs de l'image, où *max* est la valeur maximale, et où k est un reel entre 0 et 1 utilisée pour encoder une couleur. La couleur relative de l'image est ici la moyenne des couleurs présente dans l'image (on peut tester différents type de moyenne).

Répartition du travail

Ce travail peut se diviser en plusieurs grandes tâches aux intersections non vides.

- Conception et implémentation de deux structures de données, pour stocker image et arbre.
- Ecriture de *parsers* pour lire les fichiers PGM de l'image et les fichiers *arbre*.
- Implémentation de l'écriture dans des fichiers des structures arbre et image au bon format
- Ecriture d'un algorithme permettant de convertir les arbres en images.
- Ecriture de l'algorithme de conversion d'une image en arbre.
- Implémentation des différentes fonctions `uniforme()` proposées.
- Modification du code pour prendre en compte des images quelconques.

Travail attendu

Ce projet est a effectuer par groupe de 5 personnes.

Le travail a rendre se compose du code en C++ indenté et commenté et d'un rapport détaillant la phase de conception, les tests effectués, etc ...

Le tout est a rendre avant le ***vendredi 15 janvier 2010***

Une soutenance de 30 minutes aura lieu le ***mercredi 20 janvier 2010*** elle devra insister sur les différentes difficultés rencontrées et les solutions apportée.

Conseils

Prenez le temps d'effectuer une conception pertinente. Un bonne conception élimine la plupart des difficultés algorithmiques et permet une répartition facile du travail.

Répartissez-vous les tâches équitablement 1 ou 2 personnes par taches.

Ne cloisonnez pas les tâches tout le monde doit avoir une vision à peu prés claire de l'ensemble du projet.