TP2 — Compression d'images

Le but de cette séance de TP est de vous faire manipuler des arbres quaternaires. En particulier, nous nous intéresserons à fournir une méthode de compression d'images. L'idée générale à considérer est de découper l'image, d'identifier et de regrouper les zones contenant la même couleur.

Nous vous conseillons de travailler avec

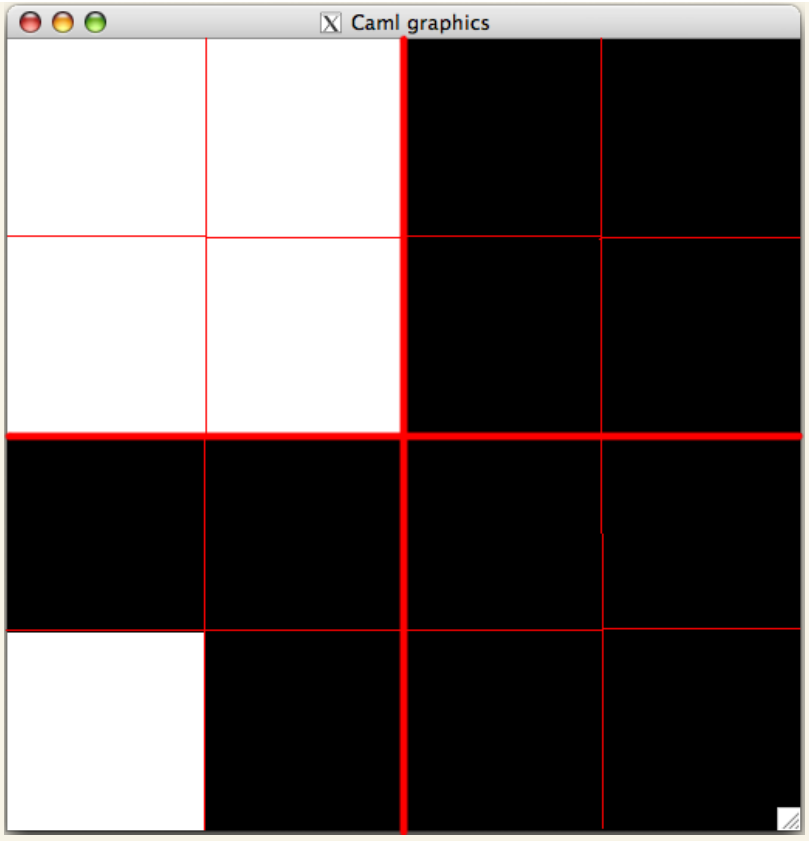
1. L'environnement Emacs et le mode Tuareg activé (installation locale sur les machines PUIO);
2. La documentation OCaml de la bibliothèque `Graphics`.

Représentation des arbres quaternaires

Pour représenter de tels types d'arbres, nous adopterons la définition de type suivante

```
type 'a t = F of 'a | N of 'a t * 'a t * 'a t * 'a t
type image_tree = int t
```

- `F(c)` représente un ou plusieurs *pixel(s)* de couleur `c`
- `N(so,se,no,ne)` représente les sous-images de la forme sud-ouest, sud-est, nord-ouest et nord-est.



L'image ci-dessus peut être représentée par un tableau de tableau d'entiers `t0` stockant un par un chacun des pixels de l'image. Dès lors, une ligne de cette matrice correspond à une colonne de pixels de l'image et la case `(t0.(i)).(j)` correspond au pixel de coordonnées `(i, j)` en partant du coin bas à gauche.

```
val t0 : int array array =
[|
  [| white ; black ; white ; white |] ;
  [| black ; black ; white ; white |] ;
  [| black ; black ; black ; black |] ;
  [| black ; black ; black ; black |]
|];;
```

Néanmoins, on se rend compte qu'il est possible de regrouper plusieurs pixels de même

couleur. Il est possible d'obtenir la forme suivante, où l'on a découpé récursivement par quatre l'image puis identifié et factorisé les zones de même couleur sous la forme d'un arbre quaternaire `a0`.

```
val a0 : image_tree =
N (
  N (
    F (white),
    F (black),
    F (black),
    F (black)),
  F (black),
  F (white),
  F (black))
```

Nous partons de l'hypothèse que `black` et `white` sont deux variables globales d'entiers que vous définierez avec `Graphics.black` et `Graphics.white`. De plus, les images seront des carrés de côtés dont la longueur est une puissance de `2`. Vous êtes vivement encouragés à utiliser les notions vues en cours : types somme, récursion, filtrage par motif... Commencez votre TP avec le squelette `TP2.ml` !

Exercice 1 — Décompression

La première étape consiste à passer des images sous la forme d'arbres quaternaires aux représentations usuelles, c'est-à-dire sous la forme d'un tableau de tableau de pixels.

1. Donner une fonction `get_pixel : int -> int -> int -> image_tree -> int` telle que `get_pixel x y longueur arbre` renvoie la couleur du pixel de coordonnées `(x, y)` dans l'image représentée par l'arbre quaternaire `arbre` de taille `longueur × longueur`

Exemple : `get_pixel 2 1 4 a0` doit renvoyer la valeur `black`

Indications : Vous effectuerez un parcours récursif sur la taille de l'image `longueur` en la divisant par deux à chaque fois et en vous déplaçant pertinamment sur un des quatre carrés découpés de votre image.

2. Déduire une fonction `image_matrix_of_tree : int -> image_tree -> int array array` telle que `image_matrix_of_tree longueur arbre` renvoie une image sous la forme de tableau de tableau d'entiers de taille `longueur × longueur` à partir de la forme d'arbre `arbre`

   Exemple : `image_matrix_of_tree 4 a0` doit renvoyer le tableau `t0`.

   Indications : Vous pouvez vous aider de la fonction `Array.init : int -> (int -> 'a) -> 'a array` telle que `Array.init n f` retourne un tableau de taille `n` où chacun des `i`-ième éléments est initialisé à `f i`.

Exercice 2 — Compression

Le seconde étape consiste désormais à passer des images sous la forme de tableaux de tableaux de pixels aux arbres quaternaires.

1. Donner une fonction `monochrome_color : int array array -> int -> int -> int -> int -> bool` telle que `monochrome_color image_matrix x y longueur couleur` renvoie `true` si dans l'image `image_matrix`, le carré de `(x, y)` à `(x + longueur, y + longueur)` est uniformément rempli de la couleur `couleur`

   Exemple : `monochrome_color t0 2 2 2 black` doit renvoyer `true`.

Indications : Vous pouvez arrêter le parcours dès lors qu'une couleur diffère de celle que vous cherchez avec le déclenchement et le rattrapage d'une exception.

2. Donner une fonction `image_tree_of_matrix : int array array -> int -> int -> int -> image_tree` telle que `image_tree_of_matrix image_matrix x y longueur` renvoie l'image sous la forme d'un arbre quaternaire à partir de l'image `image_matrix` dans le carré de `(x, y)` à `(x + longueur, y + longueur)`.

   Exemple : `image_tree_of_matrix t0 0 0 4` doit renvoyer `a0`.

   Indications : Testez si l'image admet une couleur uniforme et construisez la feuille de couleur correspondante. Sinon, divisez l'image en quatre parties avec le constructeur `N` et poursuivez la récursion.

3. Déduire une fonction `compress : int array array -> image_tree`.

Exercice 3 — Mise en pratique

Nous vous fournissons deux jeux de tests que vous pouvez exploiter avec vos implémentations.

1. `TP2_exemple_1.ml` : Trois images extraites d'un très célèbre jeu vidéo vous sont donnés. Pouvez-vous deviner de quel jeu vidéo il s'agit ?
2. `TP2_exemple_2.ml` : Une fractale de Mandelbrot vous est donné sous forme d'arbre dans `mandelbrot.quadtree`. Décompressez-la et affichez avec les fonctions `Graphics.make_image` et `Graphics.draw_image`.