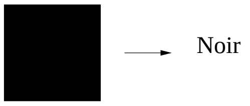
Devoir surveillé

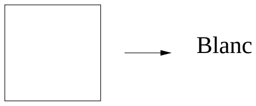
Compression d'images

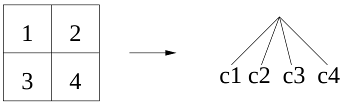
8 avril 1997 – 2 heures

On présente ici une méthode originale de représentation d'images sous forme d'arbres. Cette représentation donne une méthode de compression plus ou moins efficace et facilite certaines opérations sur les images.

On suppose les images carrées, de largeur 2^n, et bicolores (dans un but de simplification uniquement). L'idée est la suivante : si l'image est monochrome, elle se représente directement par sa couleur ; dans le cas contraire, on la divise en quatre images de même taille et on la représente par les quatres représentations correspondantes.







On se donne un type `couleur` des couleurs à deux constructeurs `Blanc` et `Noir` et un type `arbre` pour les arbres quaternaires. Les images seront des tableaux bi-dimensionnels d'éléments de type `couleur`, que l'on supposera de taille 2^n par la suite.

```
type couleur = Blanc | Noir;;

type arbre = Feuille of couleur
           | Noeud   of arbre * arbre * arbre * arbre;;

type image == couleur vect vect;;
```

1 Conversions entre les images et les arbres

Arbre correspondant à une image

☐ Ecrire une fonction `image_vers_arbre` qui prend un entier k et une image de taille $k \times k$ (on rappelle que k est de la forme 2^n) et qui rend l'arbre représentant cette image.

Image correspondant à un arbre

☐ Ecrire une fonction `remplie_carre` qui prend un tableau t, deux indices i et j, un entier k et une couleur c, et qui affecte les éléments $t_{x,y}$ pour $i \leq x \leq i+k-1$ et $j \leq y \leq j+k-1$ avec la valeur c.

☐ Ecrire une fonction `arbre_vers_image` qui prend un entier k et un arbre représentant une image de taille $k \times k$ et qui rend l'image représentée par cet arbre.

2 Conversions entre les arbres et les chaînes de caractères

Dans le but, par exemple, d'écrire cette représentation d'image dans un fichier, on se pose le problème de transformer le type `arbre` en une liste de `0` et de `1`, ainsi que le problème réciproque.

On note $code(a)$ la liste de **0** et de **1** représentant un arbre a. On choisit pour $code$ le codage suivant :

$$\begin{aligned} code(\texttt{Feuille Blanc}) &= \mathbf{00} \\ code(\texttt{Feuille Noir}) &= \mathbf{01} \\ code(\texttt{Noeud }(a_1, a_2, a_3, a_4)) &= \mathbf{1}code(a_1)code(a_2)code(a_3)code(a_4) \end{aligned}$$

Dans un soucis de simplification on va considérer ici des listes de charactères (c'est-à-dire de type `char list`) plutôt que des chaîne de caractères.

- ☐ Ecrire une fonction `arbre_vers_liste : arbre -> char liste` qui transforme un arbre en une liste de **0** et de **1** selon le codage ci-dessus.
- ☐ Ecrire une fonction `liste_vers_arbre : char list -> arbre` qui transforme une liste de **0** et de **1** en l'arbre correspondant.
- ☐ On note $f(a)$ le nombre de feuilles d'un arbre a, $n(a)$ son nombre de nœuds et $l(a)$ la longueur de la liste représentant l'arbre a. Quelle relation lie ces trois grandeurs ? Si a représente une image de taille $2^n \times 2^n$, donner un majorant de $f(a)$ et de $n(a)$ en fonction de n. En déduire le rapport maximal entre $l(a)$ et la taille de l'image (4^n).

3 Manipulation d'images sous forme d'arbres

On s'intéresse ici à des manipulations d'images (zoom, rotations, ...) effectuées non pas sur les images en tant que matrices mais directement sur leur représentation sous forme d'arbre.

Agrandissement

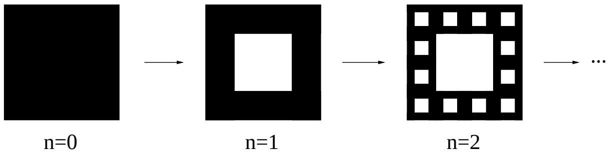
- ☐ Ecrire une fonction `zoom : arbre -> arbre` qui agrandie une image d'un facteur 2.

Rotation de 90°

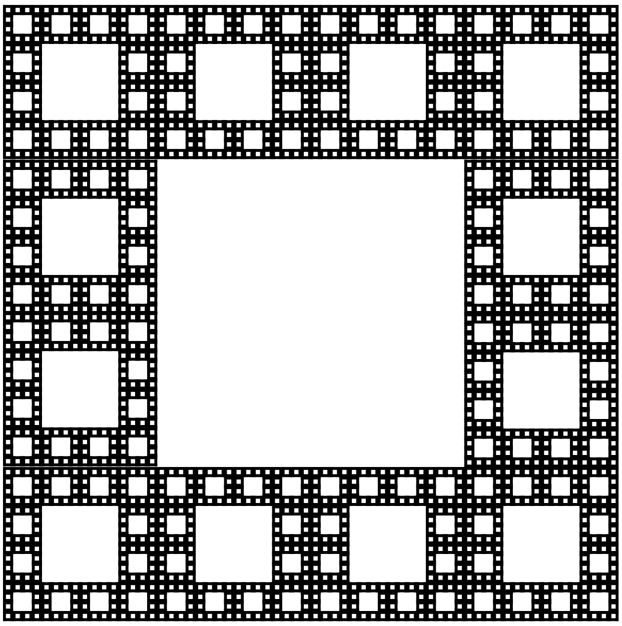
- ☐ Ecrire une fonction `rotation : arbre -> arbre` qui effectue une rotation de l'image de 90° à droite.

4 Application : dessin d'une fractale

- ☐ Ecrire une fonction `dessine_arbre` qui prend un entier k et un arbre et affiche l'image (de taille $k \times k$) représentée par cet arbre. On notera que c'est quasiment la même fonction que `arbre_vers_image`. (La fonction `fill_rect` x y l h permet de remplir une portion rectangulaire d'extrémité (x, y), de largeur l et de hauteur h).
- ☐ Ecrire une fonction `fractale : int -> arbre` qui prend un entier n et construit l'arbre correspondant à n itérations du processus suivant :



En évaluant `(dessine_arbre 256 (fractale 4))` vous devriez obtenir l'image suivante :



☐ Quelle est la complexité en temps et en espace de votre implantation de la fonction `fractale` ? Pouvez-vous faire mieux ?