# TP #5 : Représentation d'images à l'aide de `quadtrees`

Un `quadtree` est un arbre tel que tout nœud possède `0` ou `4` fils. Cette structure est souvent utilisée pour représenter des images sous forme concise.

## Organisation et cheminement du TP

Le TP s'organise en 4 parties principales

1. Traitement récursif des `quadtrees`
2. Représenter une image à l'aide d'un `quadtree`
3. Passer d'un `quadtree` à une matrice de pixels
4. Passer d'une matrice de pixels à un `quadtree`

Il s'agira de traiter (au moins) les 2 premières parties. Les parties 3 et 4 sont aussi formulées sous forme de questions. Le code solution est fourni dans le fichier `initTP5.hs` et donc consultable si besoin.

## 1. Traitement récursif des quadtrees

Un `quadtree` est un arbre tel que tout nœud possède `0` ou `4` fils. On définit le type `quadtree` dans toute sa généralité par :

```
data Qtree a = Leaf a
             | Node (Qtree a) (Qtree a) (Qtree a) (Qtree a)
        deriving (Show, Ord, Eq)
```

Pour vous familiariser avec cette notion, vous pouvez utiliser la fonction `showQtree` définie dans le fichier `initTP5.hs`

```
q1 = (Node (Node (Leaf 11) (Leaf 12) (Leaf 13) (Leaf 14)) (Leaf 2) (Leaf 3) (Leaf 4))
q2 = (Node (Leaf 1) (Leaf 2) (Node (Leaf 31) (Leaf 32) .../...  (Leaf 334)) (Leaf 34)) (Leaf 4))

> putStr (showQtree (Leaf 1))
Leaf 1
> putStr (showQtree q1)
Node
    Node Leaf 11 Leaf 12 Leaf 13 Leaf 14
    Leaf 2
    Leaf 3
    Leaf 4
> putStr (showQtree q2)
Node
    Leaf 1
    Leaf 2
    Node
        Leaf 31
        Leaf 32
        Node Leaf 331 Leaf 332 Leaf 333 Leaf 334
        Leaf 34
    Leaf 4
```

**Question.** Définir les fonctions `nbLeaf`, `nbNoeuds` et `profondeur` qui déterminent respectivement : le nombre de feuilles, le nombre de noeuds (internes) et la profondeur d'un `quadtree`.

## 2. Représentation d'une image à l'aide d'un `quadtree`

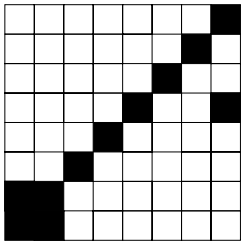
On s'intéresse aux images (carrées) possèdant $2^k \times 2^k$ pixels. Une telle image peut être représentée par un couple ($k$, `quadtree`) construit comme suit :

- si la couleur de l'image est uniformément noire (resp. blanche), alors le `quadtree` est réduit à une feuille indiquant la couleur `(Leaf N)` (resp. `(Leaf B)`).
- sinon, on partage l'image $2^k \times 2^k$ en 4 images $2^{(k-1)} \times 2^{(k-1)}$ qui seront désignées par `nw` (nord-ouest), `ne` (nord-est), `sw` (sud-ouest) et `se` (sud-est).

  La représentation `Haskell` associée est donc : `(Node nw ne sw se)`

  Chaque partie sera elle même représentée par un `quadtree`. Si l'une de ces parties est uniformément blanche ou noire, alors le `quadtree` associé est une feuille contenant la couleur sinon cette partie est elle-même divisée en 4 sous-parties, et ainsi de suite récursivement ...

On considère l'image suivante $2^k \times 2^k$ pixels qui sera partagée en 4 images $2^{(k-1)} \times 2^{(k-1)}$



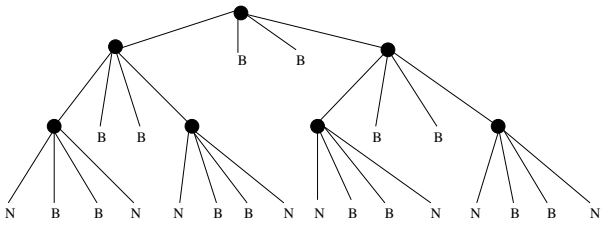
- Tous les pixels de `nw` sont blancs (idem pour `se`).
- Le `quadtree` associé est donc de la forme `(Node (Leaf B) ne sw (Leaf B))`.
- Le carré `sw` se décompose à son tour en : `(Node (Leaf B) ne' (Leaf N) (Leaf B))`.
- `ne'` se décompose à son tour en : `(Node (Leaf B) (Leaf N) (Leaf N) (Leaf B))`.
- `sw` se décompose donc en :

```
(Node (Leaf B)
      (Node (Leaf B) (Leaf N) (Leaf N) (Leaf B))
      (Leaf N)
      (Leaf B))
```

- Enfin, `ne` se décompose en :

```
(Node (Leaf B)
      (Node (Leaf B) (Leaf N) (Leaf N) (Leaf B))
      (Node (Leaf B) (Leaf N) (Leaf N) (Leaf B))
      (Node (Leaf B) (Leaf B) (Leaf B) (Leaf N)))
```

**Question.** Quelle est l'image correspondant au `quadtree` de la figure ci-dessous ?



On définit les types et fonctions suivantes :

```
data Qtree a = Leaf a
             | Node (Qtree a) (Qtree a) (Qtree a) (Qtree a)
        deriving (Show, Ord, Eq)

data Pixel = B | N
        deriving (Show, Eq)

type Image = (Int, Qtree Pixel)

type Vecteur a = [a]

type Matrice a = [Vecteur a]

showPixel :: Pixel -> Char
showPixel B = '_'
showPixel N = 'N'
```

**Questions.**

1. Pourquoi un `quadtree` seul n'est-il pas suffisant pour coder une image ?
2. Pour une image $2^k \times 2^k$, combien de pixels représente une feuille de profondeur $i$ ?
3. Compléter la définition de la fonction `(reverseVideo im)` qui permet de calculer le *négatif* d'une image représentée par un `quadtree`. Chaque pixel blanc est transformé en un pixel noir et vice-versa.

   ```
   reverseVideo :: Image -> Image
   reverseVideo (k, qt) = (k, rev qt)
          where rev (Leaf x) =
                rev (Node nw ne sw se) =
   ```

4. Définir la fonction `(transposer im)` qui permet de transposer une image représentée par un `quadtree`.

   ```
   transposer :: Image -> Image
   transposer        = (   ,   )
          where trans             =
                trans             =
   ```

## 3. Représentation matricielle d'une image

On souhaite définir la fonction `qtreeToMatrice :: Image -> (Matrice Pixel)` qui transforme une image `(k,qt)` en une matrice carrée $(2^k \times 2^k)$ de pixels.

**Exemples.**

```
q3 = (Node (Leaf B) (Node (Leaf B) (Leaf N) (Leaf B) (Leaf N)) (Leaf N) (Leaf B))

> qtreeToMatrice (2, q3)
   [ [B,B,B,N], [B,B,B,N], [N,N,B,B], [N,N,B,B] ]

> qtreeToMatrice (2, Leaf N)
   [ [N,N,N,N], [N,N,N,N], [N,N,N,N], [N,N,N,N] ]
```

**Questions.**

1. Que fait la fonction suivante ?

```
merge4 :: (Matrice Pixel)->(Matrice Pixel)->(Matrice Pixel)->(Matrice Pixel)->(Matrice Pixel)
merge4 nw ne sw se = merge2 nw ne ++ merge2 sw se
   where merge2 xss yss = [xs++ys | (xs, ys) <- zip xss yss]
```

2. Compléter les définitions suivantes :

```
qtreeToMatrice :: Image -> (Matrice Pixel)
qtreeToMatrice (k, qt) = qtm (2^k) qt

qtm ::  ->  ->
qtm k (Leaf x)           =
qtm k (Node nw ne sw se) = merge4
   where k' = div k 2
```

## 4. Représentation d'une image à l'aide d'un `quadtree`

L'objectif est de définir la fonction `matriceToQtree :: (Matrice Pixel) -> Image` qui permet de passer de la représentation matrice carrée ($2^k \times 2^k$) de pixels à la représentation `(k,qt)`

**On reprend les exemples de la Section 3.**

```
> matriceToQtree [[B,B,B,N],[B,B,B,N],[N,N,B,B],[N,N,B,B]]
   (2, Node (Leaf B) (Node (Leaf B) (Leaf N) (Leaf B) (Leaf N)) (Leaf N) (Leaf B))

> matriceToQtree [[N,N,N,N], [N,N,N,N], [N,N,N,N], [N,N,N,N]]
   (2, Leaf N)
```

On définit :

```
matriceToQtree :: (Matrice Pixel) -> Image

matriceToQtree xss = (k, regroupe (construit [[Leaf x | x <- xs] | xs <- xss]))
   where k = round (logBase 2 (fromIntegral (length (head xss))))

par4 :: (Vecteur (Qtree a)) -> (Vecteur (Qtree a)) -> (Vecteur (Qtree a))
par4      []        []      = []
par4 (nw:ne:qs) (sw:se:qs') = (Node nw ne sw se) : (par4 qs qs')

parPavesDe4 :: (Matrice (Qtree a)) -> (Matrice (Qtree a))
parPavesDe4 []          = []
parPavesDe4 (xs:ys:qss) = (par4 xs ys) : (parPavesDe4 qss)

construit :: (Matrice (Qtree a)) -> (Qtree a)
construit [[qt]] = qt
construit qss    = construit (parPavesDe4 qss)
```

**Questions.**

1. Commenter la définition de la fonction `matriceToQtree :: (Matrice Pixel) -> Image`.
2. Quel traitement réalise la fonction `construit` définie ci-dessus ? On pourra faire l'analogie avec la construction de l'arbre de HUFFMAN étudié lors du TP #4.
3. Enfin, définir la fonction `(regroupe qt)` qui permet de regrouper les carrés de même couleur.