RAPPORT

CHEVALIER Thomas 2180812 tp D td 2

Question 1:

La fonction quadtree_full crée un arbre totalement noir de hauteur k tel que n=2**k
Entrée : int n : la longueur d'un côté du carré tel que n = 2**k
Sortie : quadtree : un arbre totalement composé de feuilles Noir

Fonctionnement :

On appelle la fonction quadtree_full avec un entier n en paramètre, qui fait appel à la fonction get_height qui elle même retourne la hauteur k à partir de l'entier n.
Ensuite on construit l'arbre à partir de la hauteur k.

Lorsque k différent de 1 on appelle récursivement la fonction quadtree_full en décrémentant n de 1. Si n=1 alors nous sommes arrivé à une feuille, en l'occurrence ici, Noir.

Exemple :

```
# quadtree_full 8;;
- : quadtree =
Noeud
 (Noeud (Noeud (Feuille Noir, Feuille Noir, Feuille Noir, Feuille Noir),
   Noeud (Feuille Noir, Feuille Noir, Feuille Noir, Feuille Noir),
   Noeud (Feuille Noir, Feuille Noir, Feuille Noir, Feuille Noir),
   Noeud (Feuille Noir, Feuille Noir, Feuille Noir, Feuille Noir)),
 Noeud (Noeud (Feuille Noir, Feuille Noir, Feuille Noir, Feuille Noir),
  Noeud (Feuille Noir, Feuille Noir, Feuille Noir, Feuille Noir),
  Noeud (Feuille Noir, Feuille Noir, Feuille Noir, Feuille Noir),
  Noeud (Feuille Noir, Feuille Noir, Feuille Noir, Feuille Noir)),
 Noeud (Noeud (Feuille Noir, Feuille Noir, Feuille Noir, Feuille Noir),
  Noeud (Feuille Noir, Feuille Noir, Feuille Noir, Feuille Noir),
  Noeud (Feuille Noir, Feuille Noir, Feuille Noir, Feuille Noir),
  Noeud (Feuille Noir, Feuille Noir, Feuille Noir, Feuille Noir)),
 Noeud (Noeud (Feuille Noir, Feuille Noir, Feuille Noir, Feuille Noir),
  Noeud (Feuille Noir, Feuille Noir, Feuille Noir, Feuille Noir),
  Noeud (Feuille Noir, Feuille Noir, Feuille Noir, Feuille Noir),
  Noeud (Feuille Noir, Feuille Noir, Feuille Noir, Feuille Noir)))
```

La fonction quadtree_empty crée un arbre totalement Blanc de hauteur k tel que n=2**k

Entrée : int n : la longueur d'un côté du carré tel que n = 2**k

Sortie : quadtree : un arbre totalement composé de feuilles Blanc

Fonctionnement :

On appelle la fonction quadtree_empty avec un entier n en paramètre, qui fait appel à la fonction get_height qui elle même retourne la hauteur k à partir de l'entier n. Ensuite on construit l'arbre à partir de la hauteur k.

Lorsque k différent de 1 on appelle récursivement la fonction quadtree_empty en décrémentant n de 1. Si n=1 alors nous sommes arrivé à une feuille, en l'occurrence ici,Blanc.

Exemple :

```
# quadtree_empty 4;;
- : quadtree =
Noeud (Noeud (Feuille Blanc, Feuille Blanc, Feuille Blanc, Feuille Blanc),
 Noeud (Feuille Blanc, Feuille Blanc, Feuille Blanc, Feuille Blanc),
 Noeud (Feuille Blanc, Feuille Blanc, Feuille Blanc, Feuille Blanc),
 Noeud (Feuille Blanc, Feuille Blanc, Feuille Blanc, Feuille Blanc))
```

Question 2:

La fonction inverse prend un quadtree a et inverse les couleurs des feuilles.

Entrée : quadtree a : le quadtree a représenté une image i

Sortie : quadtree : le quadtree en retour représente l'image i' de i après application de la fonction inverse

Fonctionnement :

On parcourt récursivement notre quadtree a, si a correspond à un Noeud alors on applique la fonction inverse à chacune des feuilles de ce noeud.

Si a est de type Feuille Blanc alors a devient Feuille Noir.
Inversement, si a est de type Feuille Noir alors a devient Feuille Blanc.

Exemple :

```
# inverse (quadtree_empty 4);;
- : quadtree =
Noeud (Noeud (Feuille Noir, Feuille Noir, Feuille Noir, Feuille Noir),
 Noeud (Feuille Noir, Feuille Noir, Feuille Noir, Feuille Noir),
 Noeud (Feuille Noir, Feuille Noir, Feuille Noir, Feuille Noir),
 Noeud (Feuille Noir, Feuille Noir, Feuille Noir, Feuille Noir))
```

Question 3:

La fonction rotate prend un quadtree a et le fait tourner d un quart de tour vers la gauche.

Entrée : quadtree a : le quadtree a représenté une image i

Sortie : quadtree : le quadtree en retour represente l image i' de i apres application de la fonction rotate

Fonctionnement :

On parcourt récursivement notre quadtree a, si a correspond a un Noeud alors on applique la fonction rotate a chacun des noeuds.

On remplace les positions 1,2,3,4 par 2,3,4,1 pour chaque noeud.
Si a est de type Feuille on ne fait rien.

```
------                      ------
|1|2|                       |2|3|
------    ⇒  rotate ⇒       ------
|4|3|                       |1|4|
------                      ------
```

Initialisation pour un exemple avec des feuilles de couleurs Noir et Blanc mélangées.

```
# let exp3 = Noeud (
        Noeud (Feuille Noir, Feuille Noir, Feuille Noir, Feuille Noir),
        Noeud (Feuille Blanc, Feuille Blanc, Feuille Blanc, Feuille Blanc),
        Noeud (Feuille Blanc, Feuille Blanc, Feuille Noir, Feuille Blanc),
        Noeud (Feuille Blanc, Feuille Blanc, Feuille Blanc, Feuille Blanc) );;

val exp3 : quadtree =
  Noeud (Noeud (Feuille Noir, Feuille Noir, Feuille Noir, Feuille Noir),
   Noeud (Feuille Blanc, Feuille Blanc, Feuille Blanc, Feuille Blanc),
```

Noeud (Feuille Blanc, Feuille Blanc, Feuille Noir, Feuille Blanc),
Noeud (Feuille Blanc, Feuille Blanc, Feuille Blanc, Feuille Blanc))

Exemple :

```
# rotate exp3;;
- : quadtree =
Noeud (Noeud (Feuille Blanc, Feuille Blanc, Feuille Blanc, Feuille Blanc),
 Noeud (Feuille Blanc, Feuille Noir, Feuille Blanc, Feuille Blanc),
 Noeud (Feuille Blanc, Feuille Blanc, Feuille Blanc, Feuille Blanc),
 Noeud (Feuille Noir, Feuille Noir, Feuille Noir, Feuille Noir))
```

Question 4:

La fonction union prend 2 quadtree a et b, et on change la couleur des feuilles selon la règle.

Entrée : quadtree a, quadtree b : le quadtree a représente une image i, et le quadtree b représente une image i'

Sortie : quadtree c : le quadtree c en retour représente l'union des 2 image i et i'

Fonctionnement :

On parcourt récursivement les 2 quadtree a et b de même taille.
Si a est un type feuille, nécessairement b aussi, alors on applique la règle d'union définie par:
Blanc u X OU X u Blanc = X ; Noir u Noir = Noir.

Pour ne pas avoir l'erreur :"*Warning 8: this pattern-matching is not exhaustive.*" j'ai ajouté un dernier match pour ne pas comparer une Feuille avec un Noeud.(ce cas ne sera jamais traiter car a et b font la même taille)

Initialisation pour un exemple avec des feuilles de couleurs Noir et Blanc mélangées.

```
# let exa = Noeud (Feuille Blanc, Feuille Blanc, Feuille Noir, Feuille Noir);;
val exa : quadtree =
  Noeud (Feuille Blanc, Feuille Blanc, Feuille Noir, Feuille Noir)
# let exb = Noeud (Feuille Blanc, Feuille Noir, Feuille Noir, Feuille Blanc);;
val exb : quadtree =
  Noeud (Feuille Blanc, Feuille Noir, Feuille Noir, Feuille Blanc)
```

Exemple :

```
# union exa exb;;
```

- : quadtree =
Noeud (Feuille Blanc, Feuille Noir, Feuille Noir, Feuille Noir)

Question 5:

La fonction intersection prend 2 quadtree a et b, et on change la couleur des feuilles selon la règle.

Entrée : quadtree a, quadtree b : le quadtree a représente une image i, et le quadtree b représente une image i'

Sortie : quadtree c : le quadtree c en retour représente intersection des 2 image i et i'

Fonctionnement :

On parcourt récursivement les 2 quadtree a et b de même taille.
Si a est un type feuille, nécessairement b aussi, alors on applique la règle d'intersection définie par:
Blanc n X OU X n Blanc = Blanc ; Noir n Noir = Noir.

Pour ne pas avoir l'erreur :"*Warning 8: this pattern-matching is not exhaustive.*" j'ai ajouté un dernier match pour ne pas comparer une Feuille avec un Noeud.(ce cas ne sera jamais traiter car a et b font la même taille)

Exemple :
 # intersection exa exb;;
- : quadtree =
Noeud (Feuille Blanc, Feuille Blanc, Feuille Noir, Feuille Blanc)

Initialisation pour un exemple avec des feuilles de couleurs Noir et Blanc mélangées. (exp correspond au carré donné dans le sujet)

```
# let exp = Noeud (
            Noeud (
            Noeud (Feuille Blanc, Feuille Blanc, Feuille Blanc, Feuille Blanc),
            Noeud (Feuille Blanc, Feuille Blanc, Feuille Blanc, Feuille Blanc),
            Noeud (Feuille Blanc, Feuille Blanc, Feuille Blanc, Feuille Blanc),
            Noeud (Feuille Blanc, Feuille Blanc, Feuille Blanc, Feuille Blanc)
            ),
            Noeud (
            Noeud (Feuille Blanc, Feuille Blanc, Feuille Blanc, Feuille Blanc),
```

```
            Noeud (Feuille Blanc, Feuille Blanc, Feuille Blanc, Feuille Blanc),
            Noeud (Feuille Noir, Feuille Blanc, Feuille Blanc, Feuille Blanc),
            Noeud (Feuille Noir, Feuille Noir, Feuille Noir, Feuille Noir)
            ),
            Noeud (
            Noeud (Feuille Noir, Feuille Noir, Feuille Noir, Feuille Noir),
            Noeud (Feuille Blanc, Feuille Blanc, Feuille Blanc, Feuille Blanc),
            Noeud (Feuille Blanc, Feuille Blanc, Feuille Noir, Feuille Blanc),
            Noeud (Feuille Blanc, Feuille Blanc, Feuille Blanc, Feuille Blanc)
            ),
            Noeud (
            Noeud (Feuille Noir, Feuille Noir, Feuille Noir, Feuille Noir),
            Noeud (Feuille Noir, Feuille Noir, Feuille Noir, Feuille Noir),
            Noeud (Feuille Noir, Feuille Noir, Feuille Noir, Feuille Noir),
            Noeud (Feuille Noir, Feuille Noir, Feuille Noir, Feuille Noir)
            )
        );;

val exp : quadtree =
  Noeud
   (Noeud
        (Noeud (Feuille Blanc, Feuille Blanc, Feuille Blanc, Feuille Blanc),
        Noeud (Feuille Blanc, Feuille Blanc, Feuille Blanc, Feuille Blanc),
        Noeud (Feuille Blanc, Feuille Blanc, Feuille Blanc, Feuille Blanc),
        Noeud (Feuille Blanc, Feuille Blanc, Feuille Blanc, Feuille Blanc)),
    Noeud (Noeud (Feuille Blanc, Feuille Blanc, Feuille Blanc, Feuille Blanc),
        Noeud (Feuille Blanc, Feuille Blanc, Feuille Blanc, Feuille Blanc),
        Noeud (Feuille Noir, Feuille Blanc, Feuille Blanc, Feuille Blanc),
        Noeud (Feuille Noir, Feuille Noir, Feuille Noir, Feuille Noir)),
    Noeud (Noeud (Feuille Noir, Feuille Noir, Feuille Noir, Feuille Noir),
        Noeud (Feuille Blanc, Feuille Blanc, Feuille Blanc, Feuille Blanc),
        Noeud (Feuille Blanc, Feuille Blanc, Feuille Noir, Feuille Blanc),
        Noeud (Feuille Blanc, Feuille Blanc, Feuille Blanc, Feuille Blanc)),
    Noeud (Noeud (Feuille Noir, Feuille Noir, Feuille Noir, Feuille Noir),
        Noeud (Feuille Noir, Feuille Noir, Feuille Noir, Feuille Noir),
        Noeud (Feuille Noir, Feuille Noir, Feuille Noir, Feuille Noir),
        Noeud (Feuille Noir, Feuille Noir, Feuille Noir, Feuille Noir)))
```

Exemple :

```
# intersection exp (inverse exp);;
- : quadtree =
```

Noeud
(Noeud (Noeud (Feuille Blanc, Feuille Blanc, Feuille Blanc, Feuille Blanc),
Noeud (Feuille Blanc, Feuille Blanc, Feuille Blanc, Feuille Blanc),
Noeud (Feuille Blanc, Feuille Blanc, Feuille Blanc, Feuille Blanc),
Noeud (Feuille Blanc, Feuille Blanc, Feuille Blanc, Feuille Blanc)),
Noeud (Noeud (Feuille Blanc, Feuille Blanc, Feuille Blanc, Feuille Blanc),
Noeud (Feuille Blanc, Feuille Blanc, Feuille Blanc, Feuille Blanc),
Noeud (Feuille Blanc, Feuille Blanc, Feuille Blanc, Feuille Blanc),
Noeud (Feuille Blanc, Feuille Blanc, Feuille Blanc, Feuille Blanc)),
Noeud (Noeud (Feuille Blanc, Feuille Blanc, Feuille Blanc, Feuille Blanc),
Noeud (Feuille Blanc, Feuille Blanc, Feuille Blanc, Feuille Blanc),
Noeud (Feuille Blanc, Feuille Blanc, Feuille Blanc, Feuille Blanc),
Noeud (Feuille Blanc, Feuille Blanc, Feuille Blanc, Feuille Blanc)),
Noeud (Noeud (Feuille Blanc, Feuille Blanc, Feuille Blanc, Feuille Blanc),
Noeud (Feuille Blanc, Feuille Blanc, Feuille Blanc, Feuille Blanc),
Noeud (Feuille Blanc, Feuille Blanc, Feuille Blanc, Feuille Blanc),
Noeud (Feuille Blanc, Feuille Blanc, Feuille Blanc, Feuille Blanc)))

Question 6:

La fonction color prend 2 coordonnées (x,y), la taille de l'image i, et le quadtree qui représente l'image i, on retourne la couleur de la feuille au coordonnée (x,y).
Entrée : int*int (x,y) (on commence a compté à partir de 1!), int l : taille de l image, quadtree a : le quadtree a représente une image i.
Sortie : couleur option c : couleur du point de coordonnées (x,y) dans le quadtree a.

Les bases :

On commence a compté à partir de 1!, l'axe des abscisses (x) se situe à vertical, lors d'un déplacement vers la case du bas. Donc l'axe des ordonnées se situe à l'horizontal. (rotation du plan RxR de +90°)

Fonctionnement :

Je souhaite me reduire a un carré de 1 de côte.
Pour cela, il faut repérer où se situe le couple (x,y) dans la grille qui se décompose en 4 sous carrée (les 4 noeuds).

| 1 | 2 |

| 4 | 3 |

On s'assure que les coordonnées de x et y sont valide sinon on sort avec None.

Si le couple (x,y) se situe dans le carré 1 alors on rappel récursivement la fonction en avec le quadtree du carre 1 et divisant par 2 la taille en paramètre.
Si le couple (x,y) se situe dans le carré 2 alors on rappel récursivement la fonction en avec le quadtree du carré 2 et divisant par 2 la taille en paramètre et on décrémente y de la moitié de la taille du quadtree.
Si le couple (x,y) se situe dans le carré 3 alors on rappel récursivement la fonction en avec le quadtree du carré 3 et divisant par 2 la taille en paramètre et on décrémente x de la moitié de la taille du quadtree et y de la moitié de la taille du quadtree.
Si le couple (x,y) se situe dans le carré 4 alors on rappel récursivement la fonction en avec le quadtree du carré 4 et divisant par 2 la taille en paramètre et on décrémente x de la moitié de la taille du quadtree.

Exemple :
 # color (6,7) 8 exp;;
- : couleur option = Some Blanc
 # color (0,8) 8 exp;;
- : couleur option = None

Question 7:

La fonction modify prend un quadtree qui représente l'image i, sa taille et une fonction de profile : int -> int -> couleur -> couleur. On retourne le quadtree modifié par la fonction f.
Entrée : quadtree a : quadtree a représente une image i, int t : taille de l image i et une function f : de profile : int -> int -> couleur -> couleur.
Sortie : quadtree a : après application de la fonction f en chacun des points du quadtree.

Les bases :

On commence a compté à partir de 1!, l'axe des abscisses (x) se situe à vertical, lors d'un déplacement vers la case du bas. Donc l'axe des ordonnées se situe à l'horizontal. (rotation du plan RxR de +90°)

Fonctionnement :

Je souhaite me reduire a un carré de 1 de côte. Pour lui ensuite appliquer la fonction f.
Pour cela, il faut repérer où se situe le couple (x,y) dans la grille qui se décompose en 4 sous carrée (les 4 noeuds).

```
---------
| 1 | 2 |
---------
| 4 | 3 |
---------
```

La fonction auxiliaire prend en paramètre un a quadtree, la taille de ce quadtree et un couple de coordonnées (x,y) de type int*int.
Elle sera appelée avec le quadtree a sa taille et les coordonnées de base (1,1).

Si on se situe dans le carré 1 alors on rappel récursivement la fonction auxiliaire de modify avec comme quadtree le carré 1, on divise la taille par 2, on ne modifie pas les coordonnées de x et y.
Si on se situe dans le carré 2 alors on rappel récursivement la fonction auxiliaire de modify avec comme quadtree le carré 2, on divise la taille par 2, on modifie les coordonnées de x en lui ajoutant la taille du carré divisé par 2, et rien de plus pour y.
Si on se situe dans le carré 3 alors on rappel récursivement la fonction auxiliaire de modify avec comme quadtree le carré 3, on divise la taille par 2, on modifie les coordonnées de x en lui ajoutant la taille du carré divisé par 2, et de y en lui ajoutant la taille du carré divisé par 2.
Si on se situe dans le carré 4 alors on rappel récursivement la fonction auxiliaire de modify avec comme quadtree le carré 4, on divise la taille par 2, on modifie les coordonnées de x rien de plus pour, mais y on lui ajoutant la taille du carré divisé par 2.

Initialisation pour un exemple

```
# let modify_f_diago_noir = fun x y c ->
  if x=y then Noir
  else c;;
val modify_f_diago_noir : 'a -> 'a -> couleur -> couleur = <fun>
```

Exemple :

```
# modify (quadtree_empty 4) 4 modify_f_diago_noir;;
- : quadtree =
Noeud (Noeud (Feuille Noir, Feuille Blanc, Feuille Noir, Feuille Blanc),
 Noeud (Feuille Blanc, Feuille Blanc, Feuille Blanc, Feuille Blanc),
 Noeud (Feuille Noir, Feuille Blanc, Feuille Noir, Feuille Blanc),
 Noeud (Feuille Blanc, Feuille Blanc, Feuille Blanc, Feuille Blanc))
```

Question 8:

La fonction optimise prend un quadtree qui représente l'image i, et retourne le quadtree optimisé tel que un nœud dont les fils sont tous de la même couleur sera modifié en feuille.

Entrée : quadtree a : quadtree a représente une image i

Sortie : quadtree a : retourné un quadtree optimiser.

Fonctionnement :

Pour optimiser l'arbre je il faut repérer les cas ou les 4 feuilles sont de couleur identique et transformer le noeud en une feuille de cette couleur.
Lorsque l'on match avec un noeud alors on applique la fonction auxiliaire de optimisés.

Je n ai pas trouvé des solution pour l optimiser complètement donc je repete la fonction auxiliaire de optimise le nombre de fois qui correspond à la hauteur du quadtree. Pour trouver la hauteur du quadtree j'utilise une fonction qui compte le nombre de noeud imbriquer.

Initialisation pour un exemple

```
# let rec get_height_quadtree = fun a ->
  match a with
        | Feuille c -> 0
        | Noeud (x1, x2, x3, x4) -> 1+ get_height_quadtree x1;;
 val get_height_quadtree : quadtree -> int = <fun>
```

Exemple :

```
# optimise exp;;
```

- : quadtree =
Noeud (Feuille Blanc,
Noeud (Feuille Blanc, Feuille Blanc,
Noeud (Feuille Noir, Feuille Blanc, Feuille Blanc, Feuille Blanc),
Feuille Noir),
Noeud (Feuille Noir, Feuille Blanc,
Noeud (Feuille Blanc, Feuille Blanc, Feuille Noir, Feuille Blanc),
Feuille Blanc),
Feuille Noir)

Exemple :
 # optimise (quadtree_empty 16);;
- : quadtree = Feuille Blanc

Question 9:

La fonction quadtree_to_list prend un quadtree qui représente l'image i, et retourne une liste de bit le représentant.
Entrée : quadtree a : quadtree a représente une image i
Sortie : bit list : retourne liste de bit

Fonctionnement :

Pour transformer le quadtree en une liste de bit, on regarde regarde récursivement, si l'on tombe sur une feuille Blanc alors on ajoute à la liste Zero Zero, si la feuille est Noir alors Zero Noir.
Et si c'est un noeud alors on ajoute un à la liste et on recommence récursivement.

Exemple :
 # quadtree_to_list exp;;
- : bit list =
[Un; Un; Un; Zero; Zero; Zero; Zero; Zero; Zero; Zero; Zero; Un; Zero; Zero;
Zero; Zero; Zero; Zero; Zero; Zero; Un; Zero; Zero; Zero; Zero; Zero; Zero;
Zero; Zero; Un; Zero; Zero; Zero; Zero; Zero; Zero; Zero; Zero; Un; Un;
Zero; Zero; Zero; Zero; Zero; Zero; Zero; Zero; Un; Zero; Zero; Zero; Zero;
Zero; Zero; Zero; Zero; Un; Zero; Un; Zero; Zero; Zero; Zero; Zero; Zero;
Un; Zero; Un; Zero; Un; Zero; Un; Zero; Un; Un; Un; Zero; Un; Zero; Un;
Zero; Un; Zero; Un; Un; Zero; Zero; Zero; Zero; Zero; Zero; Zero; Zero; Un;
Zero; Zero; Zero; Zero; Zero; Un; Zero; Zero; Un; Zero; Zero; Zero; Zero;

Zero; Zero; Zero; Zero; Un; Un; Zero; Un; Zero; Un; Zero; Un; Zero; Un; Un; Zero; Un; Zero; Un; Zero; Un; Zero; Un; Un; Zero; Un; Zero; Un; Zero; Un; Zero; Un; Un; Zero; Un; Zero; Un; Zero; Un; Zero; Un; Zero; Un; Un; Zero; Un; Un; Zero; Un; Zero; Un; Zero; Un; Zero; Un]

Question 10:

Je n'ai pas réussi à réaliser la fonction.