Rapport
Devoir Maison Ocaml :
Quadtrees et compression d'image

(**Je vais réécrire ce que j'ai déjà mis en commentaire sur mon fichier dmocaml,ml**)

Question 1 :

La fonction ***quadtree_full*** prend en paramètre un nombre ***n*** (correspondant a la taille du quadtree a construire) et nous retourne un quadtree de taille n, on gere les erreurs dans un premier temps le cas ou le n est nul ou négatif on retourne une erreur et dans le cas ou n est supérieur ou égale a 1, on rentre dans la sous-fonction aux qui prend en paramètre une taille t (correspondant a la taille du quadtree-1 "n-1") et un quadtree a initialement correspondant a Feuille Noir, dans un premier cas lorsque la taille t=0 (n-1 =0 qui signifie qu'on veut construire un quadtree de taille 1) on retourne a (Feuille Noir correspondant au paramètre a), dans le deuxième cas, on appelle récursivement la fonction aux avec comme paramètre la taille t/2 ainsi que le Nœud(a,a,a,a) c'est a dire le Nœud avec "Feuille Noir * Feuille Noir * Feuille Noir * Feuille Noir" autant de fois que nécessaire.
Typage : *val quadtree_full : int -> quadtree = <fun>*

La fonction ***quadtree_empty*** prend en paramètre un nombre ***n*** (correspondant a la taille du quadtree a construire) et nous retourne un quadtree de taille n, on gère les erreurs dans un premier temps le cas ou le n est nul ou négatif on retourne une erreur et dans le cas ou n est supérieur ou egale a 1, on rentre dans la sous-fonction aux qui prend en paramètre une taille t (correspondant a la taille du quadtree-1 "n-1") et un quadtree a initialement correspondant a Feuille Blanc, dans un premier cas lorsque la taille t=0 (n-1 =0 qui signifie qu'on veut construire un quadtree de taille 1) on retourne a (Feuille Blanc correspondant au paramètre a), dans le deuxième cas, on appelle récursivement la fonction aux avec comme paramètre la taille t/2 ainsi que le Nœud(a,a,a,a) c'est a dire le Nœud avec "Feuille Blanc * Feuille Blanc * Feuille Blanc * Feuille Blanc" autant de fois que nécessaire.
Typage : *val quadtree_empty : int -> quadtree = <fun>*

Question 2 :

La fonction ***inverse*** prend en paramètre ***un quadtree*** et inverse la couleur des feuilles, et on l'appelle en récursif toutes les fois ou notre arbre possède un Nœud et en ré-exécute la fonction pour chaque quart de notre Nœud.
Typage : *val inverse : quadtree -> quadtree = <fun>*

Question 3 :

La fonction ***rotate*** prend en paramètre ***un quadtree*** et effectue une rotation d'un cran vers la gauche et on l'appelle en récursif toutes les fois ou notre arbre possède un Nœud et en ré-exécute la fonction pour chaque quart de notre Nœud pour avoir une rotation juste.
Typage : *val rotate : quadtree -> quadtree = <fun>*

Question 4 :

La fonction ***union*** prend deux paramètres ***a*** et ***b*** (a correspondant au premier quadtree) (b correspondant au deuxième quadtree) et cette fonction nous permet d'effectuer une union entre les deux arbres (selon la couleur de la feuille) et on l'appelle en récursif pour nous permettre d'appliquer la fonction sur toutes les feuilles de nos Nœuds de nos quadtree.
Typage : *val union : quadtree -> quadtree -> quadtree = <fun>*

Question 5 :

La fonction ***intersection*** prend deux paramètres ***a*** et b (a premier quadtree) (b deuxième quadtree) et cette fonction permet d'effectuer une intersection entre les deux quadtree selon la couleur de la feuille selon les "règles" imposés
Typage : *val intersection : quadtree -> quadtree -> quadtree = <fun>*

Question 6: NON Fini: *J'epere que vous allez prendre en compte (l'essai) je vous remercie d'avance,*

La fonction ***list_max*** qui me permet de retourner le maximum d'une liste ***l***,
la fonction ***compteur*** qui prend en paramètre un quadtree et qui me retourne (int) le nombre maximum de Nœuds "interne" qui se situe a l'intérieur des quarts (ceci nous permettra d'avoir la taille de notre image) une fonction ***color*** qui prend en paramètre des coordonnées ***x y*** et ***a*** un quadtree, la variable taille_image correspond a la taille totale de notre image (la longueur du coté total) la sous-fonction ***aux*** prend trois paramètres ***cx cy*** et ***un arbre quadtree***
(la fonction color semble correct mais j'ai une erreur au niveau de mes conditions a l'intérieur de aux, elle ne me permet donc pas d'afficher la bonne couleur, j'ai essayé de refaire plusieurs fois mais je n'ai pas pu régler le soucis.
Typage :

- *val liste_max : 'a list -> 'a = <fun>*
- *val compteur : quadtree -> int = <fun>*
- *val color : int -> int -> quadtree -> couleur option = <fun>*

Question 8 :

La fonction ***optimise*** prend en paramètre ***un quadtree*** et nous permet de retourner ce même quadtree optimisé c'est a dire de retourner une feuille de couleur x si notre quadtree possède un Nœud qui lui même possède quatre feuille de couleur x, et bien entendu on "ré-optimise" notre quadtree a la fin grâce a la récursivité pour nous permettre de repasser dessus dans le cas ou après la première optimisation on se retrouve avec un Nœud possédant 4 feuilles de même couleur x.
Typage : *val optimise : quadtree -> quadtree = <fun>*

Question 9 :

La fonction ***quadtree_to_list*** prend en paramètre ***un quadtree*** et nous retourne une liste de type bit list, pour cela j'ai crée une sous-fonction code qui prend en paramètre un quadtree ainsi qu'une liste, cette sous-fonction nous permet de "coder" la feuille blanc -> [Zero;Zero], la feuille Noir -> [Zero;Un] et le noeud ->[Un;...], le t dans ma sous-fonction représente le tail de notre liste(qui au départ est initialise a liste vide) et par récursivité est appelé une première fois (code q4 t) avec t vide et remonte au fur et a mesure, et a chaque fois qu'on exécute code on met le résultat dans la liste qu'on va retourner.
Typage : *val quadtree_to_list : quadtree -> bit list = <fun>*

Question 10 : NON Fini: *j'espere que vous allez prendre en compte (l'essai), je vous remercie d'avance*

La fonction **list_to_quadtree** prend en paramètre ***une liste de type bit list*** et retourne un quadtree: on procède dans un premier aux tests, c'est a dire, a si la liste est vide -> on retourne une erreur, si notre liste commence par [Zero;Zero;...] ceci signifie que les deux premiers éléments correspondent au code d'une Feuille Blanc, si elle commence par [Zero;Un;...] les deux premiers éléments correspondent au code d'une Feuille Noir, et si celle ci commence par [Un;...] il s'agit d'un début de Nœud. Je retourne dans le deuxième et troisième cas un couple (Feuille Blanc ou Noir, suite) suite étant la suite de notre liste de type bit list mis a part les deux premiers éléments (un tail)

(au départ j'ai essayé de mettre Feuille Blanc et la suite de la liste type bit list sauf que ce n'est pas possible vu que Feuille Blanc est de type quadtree et ma suite est de type bit list du coup j'ai essayé de le retourner sous forme de couple) pour le quatrième cas, le cas correspondant au Nœud je n'ai pas su comment finir ma fonction, je voulais ré-appliquer en récursif ma fonction sur la suite(le tail du Nœud) et le faire quatre fois pour (q1,q2,q3,q4) (je ne sais pas si mon raisonnement est correct vu que je n'ai pas pu résoudre mon soucis)

Les Test :

```
(*test*)
(*1*)
quadtree_full (-1);;
quadtree_full 0;;
quadtree_full 1;;
quadtree_full 4 ;;

quadtree_empty (-1);;
quadtree_empty 0;;
quadtree_empty 1;;
quadtree_empty 4 ;;

(*2*)
let quadtree_0=Feuille Noir;;
let quadtree_1 =Noeud (Feuille Noir, Feuille Noir, Feuille Blanc, Feuille Blanc);;
let quadtree_2 =Noeud (Feuille Noir, Feuille Blanc, Feuille Noir, Noeud(Feuille Blanc, Feuille Blanc, Feuille
Noir, Feuille Noir));;

inverse quadtree_0;;
inverse quadtree_1;;
inverse quadtree_2;;

(*3*)
rotate quadtree_0;;
rotate quadtree_1;;
rotate quadtree_2;;

(*4*)
let quadtree_3= Noeud (Feuille Blanc, Feuille Noir, Feuille Blanc, Feuille Noir);;

union quadtree_3 quadtree_1;;
union quadtree_2 quadtree_3;;

(*5*)
intersection quadtree_0 quadtree_1;;
intersection quadtree_2 quadtree_3;;

(*6*)
let quadtree_4 =  Noeud (Noeud (Noeud(Feuille Blanc,Feuille Blanc, Feuille Noir, Feuille Noir),Feuille
Blanc, Feuille Noir,Feuille Noir ),Feuille Blanc, Feuille Noir,Noeud(Feuille Blanc, Feuille Blanc, Feuille
Noir, Feuille Noir )  );;

compteur quadtree_4;;
color 3 4 quadtree_4;;(*ne marche pas*)

(*8*)
let quadtree_5 = Noeud(Feuille Noir, Feuille Noir,Feuille Noir,Feuille Noir);;
let quadtree_6 = Noeud(Noeud(Feuille Noir, Feuille Blanc,Feuille Blanc,Feuille Blanc), Feuille Noir,Feuille
```

```
Noir,Feuille Noir);;
let quadtree_7 = Noeud(Noeud(Feuille Blanc, Feuille Blanc,Feuille Blanc,Feuille Blanc), Feuille
Noir,Feuille Noir,Feuille Noir);;
let quadtree_8 = Noeud(Noeud(Feuille Noir, Feuille Noir, Feuille Noir, Feuille Noir), Feuille Noir, Feuille
Noir,Feuille Noir);;

optimise quadtree_5;;
optimise quadtree_6;;
optimise quadtree_7;;
optimise quadtree_8 ;;

(*9*)

quadtree_to_list quadtree_0;;
quadtree_to_list quadtree_1;;
quadtree_to_list quadtree_2;;

(*10*)
let code = [Un; Un; Zero; Un; Zero; Zero; Zero; Zero; Zero; Zero; Zero; Un; Zero; Un; Zero; Un];;
```

Projet fait par:
SAIAH Nadir, L2-Info TD2, TPB