DM1 Option Info MP/MP*: Arbres quaternaires

Faire les questions 7 à 14 du sujet CCP 2014. Les fonctions demandées devront être écrites dans un fichier .ml, et on effectuera les tests demandés ci-dessous (écrire les lignes de tests dans le fichier). On s'attachera en particulier à respecter les contraintes de l'énoncé (fonctions récursives ou faisant appel à des fonctions auxiliaires récursives). Envoyer le fichier .ml à l'adresse `cyril.lacoste@ac-rennes.fr` avant le **5 décembre 2021**.

Remarque: Vous pouvez travailler seul ou par binôme (en particulier si tout le monde n'a pas Caml chez soi), dans ce cas vous envoyez un seul fichier en précisant bien les membres du binôme dans le mail.

Tests à effectuer:

Après avoir défini le type:

```
type quater =
  |Division of int*int*int*quater*quater*quater*quater
  |Bloc of int*int*int*int ;;
```

définir les objets suivants, correspondant à la figure 4 du sujet (page 10):

```
let b11_2 = Bloc(1,1,2,0);;
let b31_1 = Bloc(3,1,1,0);;
let b41_1 = Bloc(4,1,1,2);;
let b32_1 = Bloc(3,2,1,1);;
let b42_1 = Bloc(4,2,1,3);;
let d31_2 = Division(3,1,2,b31_1,b41_1,b32_1,b42_1) ;;
let b13_2 = Bloc(1,3,2,1);;
let b33_2 = Bloc(3,3,2,2);;
let a = Division(1,1,4,b11_2,d31_2,b13_2,b33_2) ;;
```

On demande d'effectuer les tests suivants à la fin de chaque question (les mettre dans le fichier):

- Question 7 :

  ```
  scinder a ;;
  scinder b11_2 ;;
  ```

- Question 8 :

  ```
  a = fusionner b11_2 d31_2 b13_2 b33_2 ;;

  (* permet de tester si l'arbre a est égal à l'arbre fusionné de droite *)
  ```

- Question 9 :

  ```
  profondeur a ;;
  ```

- Question 10 :

```
consulter 1 4 a ;;
consulter 3 2 a ;;
consulter 4 2 a ;;
consulter 2 1 a ;;
consulter 4 4 a ;;
```

- Question 11 :

```
let a2 = peindre 2 4 0 a ;;
a = peindre 2 4 1 a2 ;;

(* on vérifie que l'on retombe bien sur a en "recoloriant" a2 *)
```

- Question 12 :

```
valider a ;;
valider (scinder b11_2) ;;
```

- Question 13 :

```
let l = sauvegarder a ;;

(* on lui donne un nom pour la réutiliser à la question suivante *)
```

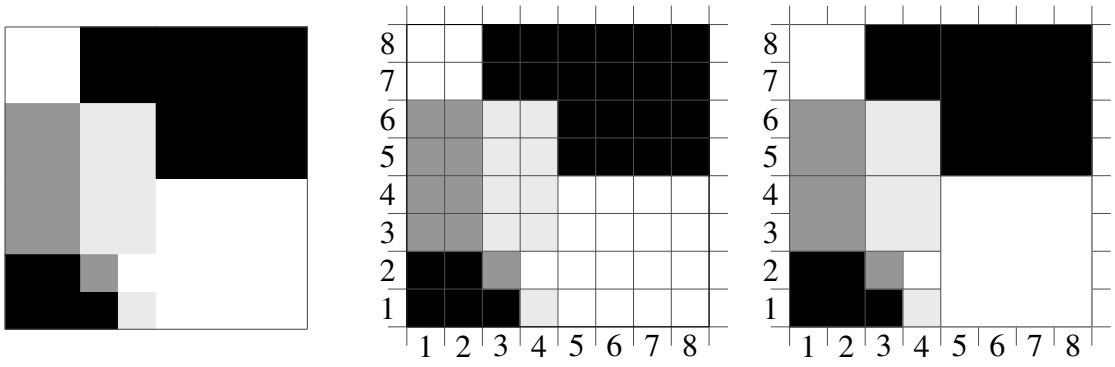
- Question 14 :

```
restaurer l = a ;;

(* pour tester si on retrouve bien l'arbre a en restaurant la liste l *)
```

2 Problème : représentation d'images par des arbres quaternaires

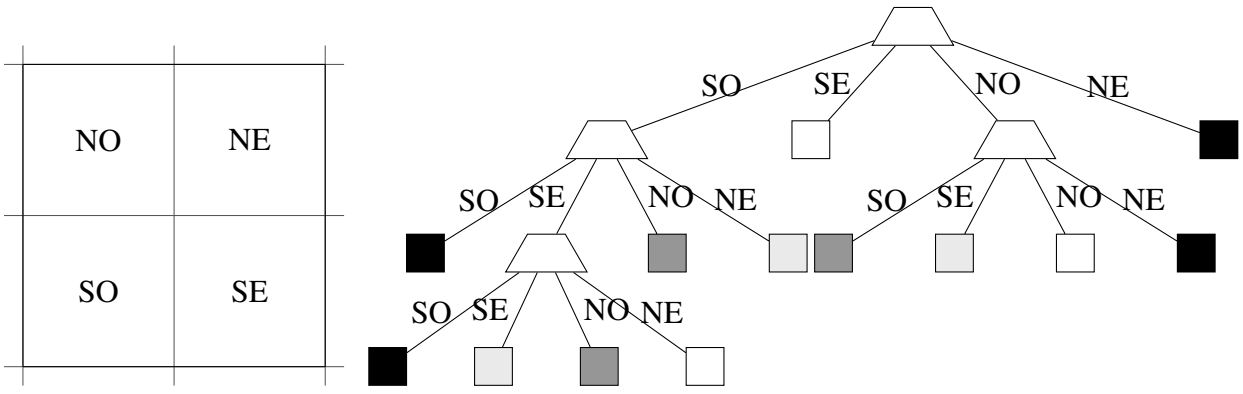
La structure d'arbre quaternaire est une extension de la structure d'arbre binaire qui permet de représenter des informations en deux dimensions. En particulier, la structure d'arbre binaire est utilisée pour représenter de manière plus compacte des images. Il s'agit de décomposer une image par dichotomie sur les deux dimensions jusqu'à obtenir des blocs de la même couleur.



(1-a) Brute (1-b) Par pixel (1-c) Par bloc

FIGURE 1: images

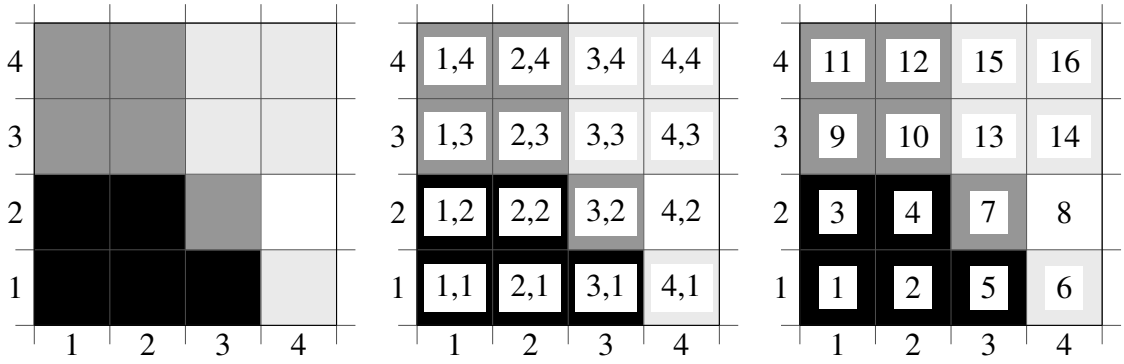
Les images des figures 1 et 2 illustrent ce principe. L'image brute (1-a) contient des carrés de différentes couleurs. Cette image est découpée uniformément en pixels (picture elements) dans l'image (1-b). Les pixels sont des carrés de la plus petite taille nécessaire. Ce découpage fait apparaître de nombreux pixels de la même couleur. Pour réduire la taille du codage, l'image (1-c) illustre un découpage dichotomique de l'image en carrés qui regroupent certains pixels de la même couleur. L'arbre quaternaire de l'image (2-b) représente les carrés contenus dans (1-c). Les étiquettes sur les fils de chaque nœud réprésentent la position géographique des fils dans le nœud : Sud-Ouest, Sud-Est, Nord-Ouest, Nord-Est comme indiqué dans (2-a).



(2-a) position géographique dans un nœud (2-b) codage de l'image de la figure 1 par un arbre quaternaire

FIGURE 2: arbre quaternaire

L'objectif de ce problème est l'étude de cette structure d'arbre quaternaire et de son utilisation pour le codage d'images en niveau de gris. Pour simplifier les programmes, nous nous limitons à des images carrées dont la longueur du côté est une puissance de 2. Les résultats étudiés se généralisent au cas des images rectangles de taille quelconque.



(3-a) Image extraite (3-b) Coordonnées (3-c) Numéros

FIGURE 3: identification des pixels

2.1 Arbres quaternaires

Déf. III.3 (Arbre quaternaire associé à une image) Un arbre quaternaire a qui représente une image carrée est composé de nœuds, qui peuvent être des blocs ou des divisions. Les ensembles des nœuds, des divisions et des blocs, de l'arbre a sont notés $\mathcal{N}(a)$, $\mathcal{D}(a)$ et $\mathcal{B}(a)$ avec $\mathcal{N}(a) = \mathcal{D}(a) \cup \mathcal{B}(a)$. Chaque nœud $n \in \mathcal{N}(a)$ contient une abscisse, une ordonnée et une taille, notées $\mathcal{X}(n)$, $\mathcal{Y}(n)$ et $\mathcal{T}(n)$, qui correspondent aux coordonnées et à la longueur du côté de la partie carrée de l'image représentée par n. Chaque bloc $b \in \mathcal{B}(a)$ contient une couleur notée $\mathcal{C}(n)$ qui correspond à la couleur de la partie carrée de l'image représentée par b. Chaque division $d \in \mathcal{D}(a)$ contient quatre fils $f_{SO}, f_{SE}, f_{NO}, f_{NE} \in \mathcal{N}(a)$. Ces fils sont indexés par la position relative de la partie carrée de l'image qu'ils représentent dans l'image représentée par la division d : Sud-Ouest, Sud-Est, Nord-Ouest et Nord-Est.

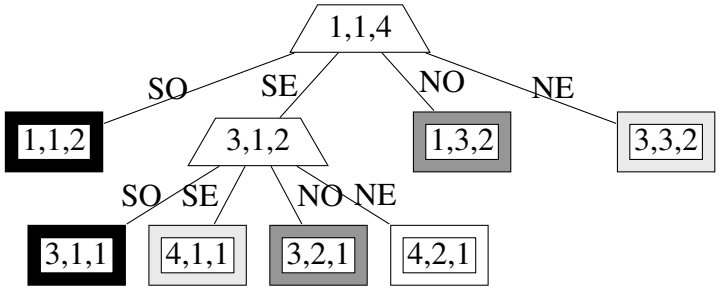


FIGURE 4: structure de données correspondant à l'image (3-a)

Exemple III.1 La figure 3 contient l'image (3-a) représentée par le sous-arbre situé au Sud-Ouest de l'arbre (2-b) page 9 ainsi que l'indication (3-b) des coordonnées de chaque point de cette image. Elle contient également la technique de numérotation des points exploitée dans la section 2.3 (page 13). La figure 4 (ci-dessus) contient l'arbre qui représente cette image dont les blocs et les divisions ont été annotés avec les coordonnées, tailles et couleurs.

Déf. III.4 (Profondeur d'un arbre quaternaire) La profondeur d'un bloc $b \in \mathcal{B}(a)$ d'un arbre quaternaire a est égale au nombre de divisions qui figurent dans la branche conduisant de la racine de a au bloc b. La profondeur d'un arbre a est le maximum des profondeurs de ses blocs $b \in \mathcal{B}(a)$.

Exemple III.2 La profondeur de l'arbre de la figure 2-b (page 9) est 3. La profondeur de l'arbre de la figure 4 (ci-dessus) est 2.

Déf. III.5 (Arbre quaternaire valide) Un arbre quaternaire a est valide, si et seulement si :

- les tailles, abscisses et ordonnées de chaque nœud $n \in \mathcal{N}(a)$ sont strictement positives ;
- les tailles des quatre fils $f_{SO}, f_{SE}, f_{NO}, f_{NE}$ de chaque division $d \in \mathcal{D}(a)$ sont identiques et égales à la moitié de la taille de d ;
- les abscisses et ordonnées des fils de chaque division $d \in \mathcal{D}(a)$ sont cohérentes avec la position géographique de chaque fils $f_{SO}, f_{SE}, f_{NO}, f_{NE}$ et avec l'abscisse, l'ordonnée et la taille de la division d ;
- au moins deux des quatre fils $f_{SO}, f_{SE}, f_{NO}, f_{NE}$ de chaque division $d \in \mathcal{D}(a)$ contiennent des blocs de couleurs différentes.

2.2 Représentation en CaML

Un arbre quaternaire est représenté par le type `quater`. La position d'un sous-arbre dans un nœud est représentée par le type énuméré `position` contenant les valeurs `SO`, `SE`, `NO` et `NE` (représentant respectivement les sous-arbres situés au Sud-Ouest, Sud-Est, Nord-Ouest et Nord-Est d'une division). Les définitions en CaML de ces types sont :

```
type quater =
  | Division of int * int * int * quater * quater * quater * quater
  | Bloc of int * int * int * int;;
type position = SO | SE | NO | NE;;
```

Dans l'appel `Bloc(x,y,t,c)` qui construit un arbre quaternaire dont la racine est un bloc, les paramètres `x` et `y` sont les coordonnées du point situé en bas à gauche de l'image représentée par ce bloc, `t` est la longueur du côté de l'image carrée représentée par ce bloc et `c` est la couleur des points de l'image représentée par ce bloc.
Dans l'appel `Division(x,y,t,so,se,no,ne)` qui construit un arbre quaternaire dont la racine est une division, les paramètres `x` et `y` sont les coordonnées du point situé en bas à gauche de l'image représentée par cette division, `t` est la longueur du côté de l'image carrée représentée par cette division, `so`, `se`, `no` et `ne` sont quatre arbres quaternaires qui sont les quatre parties de la sub-division de l'image représentée par cette division. Celles-ci sont respectivement, `so` la partie en bas à gauche (Sud-Ouest), `se` la partie en bas à droite (Sud-Est), `no` la partie en haut à gauche (Nord-Ouest), `ne` la partie en haut à droite (Nord-Est).

Exemple III.3 En associant les valeurs `0`, `1`, `2`, `3` aux couleurs noir, gris foncé, gris clair et blanc, l'expression suivante :

```
let b11_2 = Bloc( 1, 1, 2, 0) in (* SO racine *)
let b31_1 = Bloc( 3, 1, 1, 0) in (* SO du SE racine *)
let b41_1 = Bloc( 4, 1, 1, 2) in (* SE du SE racine *)
let b32_1 = Bloc( 3, 2, 1, 1) in (* NO du SE racine *)
let b42_1 = Bloc( 4, 2, 1, 3) in (* NE du SE racine *)
let d31_2 = (* SE racine *)
  Division( 3, 1, 2, b31_1, b41_1, b32_1, b42_1) in
let b13_2 = Bloc( 1, 3, 2, 1) in (* NO racine *)
let b33_2 = Bloc( 3, 3, 2, 2) in (* NE racine *)
  Division( 1, 1, 4, b11_2, d31_2, b13_2, b33_2) (* racine *)
```

est alors associée à l'arbre quaternaire représenté graphiquement sur la figure 4 (page 10).

2.2.1 Scission d'un arbre quaternaire

Question III.7 *Ecrire en CaML une fonction* `scinder` *de type* `quater -> quater` *telle que l'appel* `(scinder a)` *sur un arbre quaternaire valide* `a` *renvoie, soit un arbre quaternaire identique à l'arbre* `a` *si la racine de celui-ci est une division, soit un arbre quaternaire dont la racine est une division contenant quatre blocs de même couleur identique à celle du bloc à la racine de l'arbre* `a`*. Les coordonnées et les tailles des blocs et de la division doivent être cohérentes. Toutes les contraintes de validité de l'arbre renvoyé doivent être satisfaites sauf celle concernant les couleurs.*

2.2.2 Fusion d'arbres quaternaires

Question III.8 *Ecrire en CaML une fonction* `fusionner` *de type* `quater -> quater -> quater -> quater -> quater` *telle que l'appel* `(fusionner so se no ne)` *sur quatre arbres quaternaires valides* `so`*,* `se`*,* `no` *et* `ne` *renvoie un arbre quaternaire valide codant l'image dont les parties Sud-Ouest, Sud-Est, Nord-Ouest et Nord-Est sont codées par* `so`*,* `se`*,* `no` *et* `ne`*. Les abscisses, ordonnées et tailles de* `so`*,* `se`*,* `no` *et* `ne` *sont cohérentes avec leur position dans l'image représentée par le résultat renvoyé.*

2.2.3 Calcul de la profondeur d'un arbre quaternaire

Question III.9 *Ecrire en CaML une fonction* `profondeur` *de type* `quater -> int` *telle que l'appel* `(profondeur a)` *sur un arbre quaternaire valide* `a` *renvoie la profondeur de l'arbre* `a`*. L'algorithme utilisé ne devra parcourir qu'une seule fois l'arbre* `a`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

2.2.4 Consulter la couleur d'un point dans un arbre quaternaire

Question III.10 *Ecrire en CaML une fonction* `consulter` *de type* `int -> int -> quater -> int` *telle que l'appel* `(consulter x y a)` *sur un point d'abscisse* `x` *et d'ordonnée* `y` *et sur un arbre quaternaire valide* `a` *tel que le point d'abscisse* `x` *et d'ordonnée* `y` *soit contenu dans l'image représentée par l'arbre* `a`*, renvoie la couleur du point d'abscisse* `x` *et d'ordonnée* `y` *dans l'image représentée par l'arbre* `a`*. L'algorithme utilisé ne devra parcourir qu'une seule fois l'arbre* `a`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

2.2.5 Peindre un point dans un arbre quaternaire

Question III.11 *Ecrire en CaML une fonction* `peindre` *de type* `int -> int -> int -> quater -> quater` *telle que l'appel* `(peindre x y c a)` *sur un point d'abscisse* `x` *et d'ordonnée* `y`*, une couleur* `c` *et un arbre quaternaire valide* `a` *renvoie un arbre quaternaire valide. Le point de coordonnées* `x` *et* `y` *doit être contenu dans l'image représentée par l'arbre* `a`*. L'arbre renvoyé représente une image contenant les mêmes couleurs que l'image représentée par l'arbre* `a` *sauf pour le point de coordonnées* `x` *et* `y` *dont la couleur sera* `c`*. L'algorithme utilisé ne devra parcourir qu'une seule fois l'arbre* `a`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

2.2.6 Validation d'un arbre quaternaire

Question III.12 *Ecrire en CaML une fonction* `valider` *de type* `quater -> bool` *telle que l'appel* `(valider a)` *sur un arbre quaternaire* `a` *renvoie la valeur* `true` *si l'arbre* `a` *est valide, c'est-à-dire si* VAQ(a) *et la valeur* `false` *sinon. L'algorithme utilisé ne devra parcourir qu'une seule fois l'arbre* `a`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

2.3 Sauvegarde et restauration

Pour sauvegarder dans un fichier les couleurs contenues dans un arbre quaternaire valide, celles-ci sont rangées dans une séquence triée selon la position des points dans l'arbre. L'ordre choisi permet de restaurer l'arbre efficacement. La figure (3-c) (page 10) indique l'ordre dans lequel les points doivent être sauvegardés. La séquence manipulée contiendra les couleurs et les numéros associés à la position de chaque couleur dans l'arbre.

2.3.1 Codage en CaML

Une séquence de positions et de couleurs est représentée par le type *sequence* dont la définition est :

```
type sequence == (int * int) list;;
```

La position figure avant la couleur associée dans la séquence.

Exemple III.4 En associant les valeurs `0`, `1`, `2`, `3` aux couleurs noir, gris foncé, gris clair et blanc, la sauvegarde de l'image de la figure (3-c) (page 10) produit la séquence :

```
[(1,0); (2,0); (3,0); (4,0); (5,0); (6,2); (7,1); (8,3);
 (9,1); (10,1); (11,1); (12,1); (13,2); (14,2); (15,2); (16,2)]
```

2.3.2 Sauvegarde d'un arbre quaternaire

Question III.13 *Ecrire en CaML une fonction* `sauvegarder` *de type* `quater -> sequence` *telle que l'appel* `(sauvegarder a)` *sur un arbre quaternaire valide* `a` *renvoie une séquence triée contenant les mêmes couleurs à la même position que l'arbre* `a`*. L'algorithme utilisé ne devra parcourir qu'une seule fois l'arbre* `a` *et ne devra pas reparcourir la (ou les) séquence(s) créée(s) en dehors de leur création. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

2.3.3 Restauration d'un arbre quaternaire

Question III.14 *Ecrire en CaML une fonction* `restaurer` *de type* `sequence -> quater` *telle que l'appel* `(restaurer s)` *sur une séquence valide* `s` *renvoie un arbre quaternaire valide contenant exactement les points contenus dans la séquence* `s`*. L'algorithme utilisé ne devra parcourir qu'une seule fois la séquence* `s` *et ne devra pas reparcourir les arbres quaternaires créés en dehors de leur création. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*