Concours Commun INP (CCINP), France

Session 2014, filière MP

Informatique

Durée : 3 heures

Ce que traite le sujet

- Partie I : logique et calcul des propositions
- Partie II : automates et langages
- Partie III : algorithmique et programmation en CaML
- Partie III : algorithmique et programmation en PASCAL

Sujet reproduit sans retouche à partir de la page suivante.

Résumé des parties, chapitres de cours liés et autres années :
prepahelp.com/annales/ccinp/mp/2014/info

Prepa Help ne revendique aucun droit sur le sujet, propriété de l'organisateur du concours.

CONCOURS COMMUNS POLYTECHNIQUES

EPREUVE SPECIFIQUE - FILIERE MP

INFORMATIQUE

Durée : 3 heures

N.B. : Le candidat attachera la plus grande importance à la clarté, à la précision et à la concision de la rédaction. Si un candidat est amené à repérer ce qui peut lui sembler être une erreur d'énoncé, il le signalera sur sa copie et devra poursuivre sa composition en expliquant les raisons des initiatives qu'il a été amené à prendre.

Les calculatrices sont interdites

PREAMBULE : les trois parties qui composent ce sujet sont indépendantes et peuvent être traitées par les candidats dans un ordre quelconque.

Pour les candidats ayant utilisé le langage CaML dans le cadre des enseignements d'informatique, la partie III (Algorithmique et programmation en CaML) se situe en page 7.

Pour les candidats ayant utilisé le langage PASCAL dans le cadre des enseignements d'informatique, la partie III (Algorithmique et programmation en PASCAL) se situe en page 14.

Partie I : logique et calcul des propositions

Dans une association de jeunes détectives, les membres s'entraînent à résoudre des problèmes logiques. Ceux-ci respectent les règles suivantes : « Lors d'une conversation, un même membre aura un comportement constant : il dira toujours la vérité, ou ne dira jamais la vérité. » et « Une conversation ne doit pas être absurde. »

Question I.1 *Soient n membres intervenant dans une même conversation. Chaque membre est représenté par une variable propositionnelle M_i avec $i \in [1 \cdots n]$ qui représente le fait que ce membre dit, ou ne dit pas, la vérité. Chaque membre fait une seule déclaration représentée par la variable propositionnelle D_i. Représenter le respect des règles dans cette conversation sous la forme d'une formule du calcul des propositions dépendant des variables M_i et D_i avec $i \in [1 \cdots n]$.*

Vous assistez à deux conversations sur la véracité des déclarations de deux groupes de trois membres de cette association.
Nous nommerons A, B et C les participants de la première conversation.

A : « Les seuls qui disent la vérité ici sont C et moi. »

B : « C ne dit pas la vérité. »

C : « Soit B dit la vérité. Soit A ne dit pas la vérité. »

Nous noterons A, B et C les variables propositionnelles associées au fait que A, B et C disent respectivement la vérité.
Nous noterons D_A, D_B et D_C les formules du calcul des propositions associées respectivement aux déclarations de A, B et C dans la conversation.

Question I.2 *Représenter les déclarations de la première conversation sous la forme de formules du calcul des propositions D_A, D_B et D_C dépendant des variables A, B et C.*

Question I.3 *En utilisant le calcul des propositions (résolution avec les tables de vérité), déterminer si A, B et C disent, ou ne disent pas, la vérité.*

Nous nommerons D, E et F les participants de la seconde conversation.

D : « Personne ne doit croire F. »

E : « D et F disent toujours la vérité. »

F : « E a dit la vérité. »

Nous noterons D, E et F les variables propositionnelles associées au fait que D, E et F disent respectivement la vérité.
Nous noterons D_D, D_E et D_F les formules du calcul des propositions associées respectivement aux déclarations de D, E et F dans la seconde conversation.

Question I.4 *Représenter les déclarations de la seconde conversation sous la forme de formules du calcul des propositions D_D, D_E et D_F dépendant des variables D, E et F.*

Question I.5 *En utilisant le calcul des propositions (résolution avec les formules de De Morgan), déterminer si D, E et F disent, ou ne disent pas, la vérité.*

Partie II : automates et langages

Le but de cet exercice est de prouver qu'un homomorphisme de langage préserve la structure de langage régulier : l'image et l'image réciproque d'un langage régulier par un homomorphisme de langage sont des langages réguliers. Pour simplifier les preuves, nous nous limiterons à des homomorphismes Λ-libres. Les résultats étudiés s'étendent au cas des homomorphismes quelconques.

1 Homomorphisme de langage

Déf. II.1 (Alphabets, mots, langages) Un alphabet X est un ensemble de symboles. Un mot $x_1 \dots x_n$ de taille n sur X est une séquence de taille n de symboles $x_i \in X$. Λ est le symbole représentant le mot vide ($\Lambda \notin X$). $X^\star$ est l'ensemble contenant Λ et tous les mots sur X. L'opérateur binaire $\cdot$ est tel que : $\forall x \in X, \forall m \in X^\star, x \cdot m \in X^\star$. Un mot $x_1 \dots x_n$ de taille n s'écrit de manière unique $x_1 \cdot (x_2 \cdot (\dots \cdot (x_n \cdot \Lambda) \dots))$. Un langage sur X est un sous-ensemble de $X^\star$. L'opérateur binaire associatif sur $X^\star$ de concaténation des mots $\centerdot$ admet Λ comme élément neutre.

Déf. II.2 (Homomorphisme de langage) Soient deux alphabets X et Y, un homomorphisme de langage h est une application de domaine $X^\star$ et co-domaine $Y^\star$ qui préserve le mot vide et l'opérateur de concaténation des mots, c'est-à-dire :

$$h(\Lambda) = \Lambda$$
$$\forall m_1, m_2 \in X^\star, h(m_1 \centerdot m_2) = h(m_1) \centerdot h(m_2)$$

Déf. II.3 (Homomorphisme Λ-libre) Soit un alphabet X, soit h un homomorphisme de langage de domaine $X^\star$, h est Λ-libre, si et seulement si :

$$\forall m \in X^\star, m \neq \Lambda \implies h(m) \neq \Lambda$$

Déf. II.4 (Image d'un langage par un homomorphisme) Soit un alphabet X, soit un homomorphisme de langage h de domaine $X^\star$, soit $L_X \subseteq X^\star$ un langage sur X, l'image de L_X par h est définie par :

$$h(L_X) = \{h(m) \mid m \in L_X\}$$

Déf. II.5 (Image réciproque d'un langage par un homomorphisme) Soit un alphabet Y, soit un homomorphisme de langage h de co-domaine $Y^\star$, soit $L_Y \subseteq Y^\star$ un langage sur Y, l'image réciproque de L_Y par h est définie par :

$$h^{-1}(L_Y) = \{m \mid h(m) \in L_Y\}$$

Nous décomposons l'étude de la préservation de la structure de langage régulier par les homomorphismes de langage en plusieurs étapes qui reposent sur une restriction des homomorphismes de langage à un alphabet.

Déf. II.6 (Extension de Kleene) Soient deux alphabets X et Y, soit une application h de domaine X et co-domaine $Y^\star$, nous notons $\widehat{h}$ l'extension au domaine $X^\star$ définie par :

$$\widehat{h}(\Lambda) = \Lambda$$
$$\forall x \in X, m \in X^\star, \widehat{h}(x \cdot m) = h(x) \centerdot \widehat{h}(m)$$

Question II.1 *Soient deux alphabets X et Y, soit une application h de domaine X et co-domaine $Y^\star \setminus \{\Lambda\}$, montrer que $\widehat{h}$ est un homomorphisme de langage Λ-libre.*

Question II.2 *Soient deux alphabets X et Y, soit un homomorphisme de langage h de domaine $X^\star$ et co-domaine $Y^\star$, soit $h_{|X}$ sa restriction au domaine X, montrer que $\widehat{h_{|X}} = h$.*

2 Langages et expressions régulières

Pour montrer que l'image d'un langage régulier par un homomorphisme de langage est un langage régulier, nous exploitons la définition des langages réguliers sous la forme d'expressions régulières.

Déf. II.7 (Expression régulière) Soit un alphabet X, une expression régulière sur X est construite à partir des constantes $\emptyset$, Λ et $a \in X$, de l'opérateur unaire de répétition $\star$ et des opérateurs binaires associatifs de concaténation $\centerdot$ et de choix $+$ (qui est également commutatif). Les opérateurs ont les priorités croissantes suivantes : $+$, $\centerdot$ et $\star$.

Exemple II.1 (Expression régulière) Soit l'alphabet $X = \{a, b\}$, l'expression suivante est une expression régulière sur X : $a \centerdot (b \centerdot a \centerdot \Lambda + a \centerdot b \centerdot a \centerdot \Lambda)^\star$. Comme indiqué dans la définition des mots, pour simplifier l'écriture, nous n'utilisons pas explicitement l'opérateur $\centerdot$ et Λ. L'expression précédente sera donc notée $a(ba + aba)^\star$.

Déf. II.8 (Langage spécifié par une expression régulière) Soit l'alphabet X, soit e une expression régulière sur X, le langage régulier $L(e) \subseteq X^\star$ spécifié par e est défini par :

$$
\begin{array}{l}
L(\emptyset) = \{\} \\
L(\Lambda) = \{\Lambda\} \\
L(a) = \{a\} \text{ si } a \in X \\
L(e_1 + e_2) = L(e_1) \cup L(e_2) \\
L(e_1 \centerdot e_2) = \{m_1 \centerdot m_2 | m_1 \in L(e_1), m_2 \in L(e_2)\} \\
L(e^\star) = L(e)^\star
\end{array}
$$

Déf. II.9 (Homomorphisme d'expression régulière) Soient deux alphabets X et Y, soit h une application de domaine X et co-domaine $Y^\star$, soit e une expression régulière sur X, nous notons $\widetilde{h}$ l'application, dont le domaine est l'ensemble des expressions régulières sur X, définie par :

$$
\begin{array}{l}
\widetilde{h}(\emptyset) = \emptyset \\
\widetilde{h}(\Lambda) = \Lambda \\
\widetilde{h}(a) = h(a) \text{ si } a \in X \\
\widetilde{h}(e_1 + e_2) = \widetilde{h}(e_1) + \widetilde{h}(e_2) \\
\widetilde{h}(e_1 \centerdot e_2) = \widetilde{h}(e_1) \centerdot \widetilde{h}(e_2) \\
\widetilde{h}(e^\star) = \widetilde{h}(e)^\star
\end{array}
$$

Question II.3 *Soient les alphabets $X = \{a, b\}$ et $Y = \{0, 1\}$, soit l'application h de domaine X et co-domaine $Y^\star$ définie par $h(a) = 10$ et $h(b) = 0$, calculer $\widetilde{h}$ appliquée sur l'expression régulière de l'exemple II.1 (ci-dessus).*

Question II.4 *Soient deux alphabets X et Y, soit h une application de domaine X et co-domaine $Y^\star$, soit e une expression régulière sur X, montrer que $\widetilde{h}(e)$ est une expression régulière sur Y.*

Question II.5 *Soient deux alphabets X et Y, soit h une application de domaine X et co-domaine $Y^\star$, soit e une expression régulière sur X, montrer que $L(\widetilde{h}(e)) = \widehat{h}(L(e))$.*

Question II.6 *Soient deux alphabets X et Y, soit h un homomorphisme de langage Λ-libre de domaine $X^\star$ et co-domaine $Y^\star$, soit L_X un langage régulier sur X, montrer que $h(L_X)$ est un langage régulier sur Y.*

3 Langages réguliers et automates finis

Pour montrer que l'image réciproque d'un langage régulier par un homomorphisme de langage est un langage régulier, nous exploitons la définition des langages réguliers sous la forme d'automates finis. Pour simplifier les preuves, nous nous limiterons au cas des automates finis semi-indéterministes (automates indéterministes sans transition arbitraire sur ϵ). Les résultats étudiés s'étendent au cadre des automates finis quelconques.

Déf. II.10 (Automate fini semi-indéterministe) Soit l'alphabet X, un automate fini semi-indéterministe sur X est un quintuplet $A = (Q, X, I, T, \delta)$ composé de :

- un ensemble fini d'états : Q ;
- un ensemble d'états initiaux : $I \subseteq Q$;
- un ensemble d'états terminaux : $T \subseteq Q$;
- une fonction de transition confondue avec son graphe : $\delta \subseteq Q \times X \mapsto \mathcal{P}(Q)$.

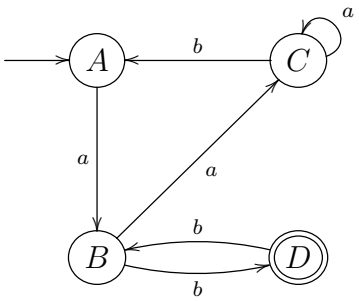
Pour une transition $\delta(o, x) = \{d_1, \ldots, d_n\}$ donnée, nous appelons o l'origine de la transition, x l'étiquette de la transition et d_1, …, d_n les différentes destinations de la transition.
Les automates peuvent être représentés par un schéma suivant les conventions :

- les valeurs de la fonction de transition δ sont représentées par un graphe orienté dont les nœuds sont les états et les arêtes sont les transitions ;
- tout état q est entouré d'un cercle (q) ;
- tout état initial i est désigné par une flêche $\longrightarrow (i)$;
- tout état terminal t est entouré d'un second cercle $((t))$;
- toute arête étiquetée par le symbole $x \in X$ va de l'état o à l'état d si et seulement si $d \in \delta(o, x)$.

Exemple II.2 L'automate $\mathcal{E} = (Q, X, I, T, \delta)$ avec :

$$
\begin{array}{l}
Q = \{A, B, C, D\} \\
X = \{a, b\} \\
I = \{A\} \\
T = \{D\} \\
\quad \delta(A, a) = \{B\} \quad \delta(B, a) = \{C\} \\
\quad \delta(B, b) = \{D\} \quad \delta(C, a) = \{C\} \\
\quad \delta(C, b) = \{A\} \quad \delta(D, b) = \{B\}
\end{array}
$$

est représenté par le graphe suivant :



Déf. II.11 (Fermeture réflexive et transitive) Soit (Q, X, I, T, δ) un automate fini semi-indéterministe, $\delta^\star \subseteq Q \times X^\star \mapsto \mathcal{P}(Q)$ la fermeture réflexive et transitive de δ sur $X^\star$ est définie par :

$$\forall q \in Q,\ \delta^\star(q, \Lambda) = \{q\}$$

$$\left\{ \begin{array}{l} \forall x \in X, \\ \forall m \in X^\star, \\ \forall o \in Q, \\ \forall d \in Q, \end{array} \right. \quad d \in \delta^\star(o, x \cdot m) \Leftrightarrow \exists q \in Q,\ (q \in \delta(o, x) \wedge d \in \delta^\star(q, m))$$

Déf. II.12 (Langage reconnu par un automate) Soit $\mathcal{A} = (Q, X, I, T, \delta)$ un automate fini semi-déterministe, $L(\mathcal{A})$ le langage régulier sur X reconnu par $\mathcal{A}$ est défini par :

$$L(\mathcal{A}) = \{m \in X^\star \mid \exists i \in I, \exists t \in T,\ t \in \delta^\star(i, m)\}$$

Question II.7 *Donner, sans justification, l'expression régulière ou ensembliste représentant le langage sur $X = \{a, b\}$ reconnu par l'automate $\mathcal{E}$ de l'exemple II.2 (page 5).*

Question II.8 *Soit (Q, X, I, T, δ) un automate fini semi-déterministe, montrer que :*

$$\forall o, d \in Q, \forall m_1, m_2 \in X^\star, d \in \delta^\star(o, m_1 \centerdot m_2) \Leftrightarrow \exists q \in Q, (q \in \delta^\star(o, m_1) \wedge d \in \delta^\star(q, m_2)$$

L'image réciproque du langage reconnu par un automate fini semi-indéterministe par un homomorphisme de langage peut être obtenue par transformation de cet automate selon la définition suivante.

Déf. II.13 (Image réciproque d'un automate fini) Soient X et Y deux alphabets, soit h une application de X dans $Y^\star$, soit $\mathcal{A} = (Q, Y, I, T, \delta)$ un automate fini semi-indéterministe, l'automate $\mathcal{B} = \widehat{h}^{-1}(\mathcal{A})$, image réciproque de l'automate $\mathcal{A}$, est défini par :

$$\begin{array}{l} \mathcal{B} = (Q, X, I, T, \delta_{\widehat{h}^{-1}}) \\ \forall x \in X,\ \forall o \in Q,\ \forall d \in Q,\ d \in \delta_{\widehat{h}^{-1}}(o, x) \Leftrightarrow o \in \delta^\star(d, \widehat{h}(x)) \end{array}$$

Question II.9 *Soient deux alphabets $X = \{a, b\}$ et $Y = \{0, 1\}$ soit h une application de domaine Y et co-domaine $X^\star$ telle que $h(0) = ab$ et $h(1) = b$, construire l'automate $\widehat{h}^{-1}(\mathcal{E})$ pour l'automate $\mathcal{E}$ de l'exemple II.2.*

Question II.10 *Caractériser le langage reconnu par $\widehat{h}^{-1}(\mathcal{E})$ par une expression régulière ou ensembliste. Comparer le langage reconnu par $\widehat{h}(\widehat{h}^{-1}(\mathcal{E}))$ avec le langage reconnu par $\mathcal{E}$.*

Question II.11 *Soit (Q, X, I, T, δ) un automate fini semi-indéterministe, soit une application h de domaine X et co-domaine $Y^\star$, montrer que :*

$$\forall m \in X^\star,\ \forall o \in Q,\ \forall d \in Q,\ d \in \delta^\star_{\widehat{h}^{-1}}(o, m) \Leftrightarrow d \in \delta^\star(o, \widehat{h}(m))$$

Question II.12 *Soit un automate fini semi-indéterministe $\mathcal{A}$, soit une application h de domaine X et co-domaine $Y^\star$, quelle relation liant les langages reconnus par $\mathcal{A}$ et $\widehat{h}^{-1}(\mathcal{A})$ peut-on déduire des questions précédentes ?*

Question II.13 *Soient deux alphabets X et Y, soit h un homomorphisme de langage Λ-libre de domaine $X^\star$ et co-domaine $Y^\star$, soit L_Y un langage régulier sur Y, montrer que $h^{-1}(L_Y)$ est un langage régulier sur X.*

Partie III : algorithmique et programmation en CaML

Cette partie doit être traitée par les étudiants qui ont utilisé le langage CaML dans le cadre des enseignements d'informatique. Les fonctions écrites devront être récursives ou faire appel à des fonctions auxiliaires récursives. Elles ne devront pas utiliser d'instructions itératives (c'est-à-dire `for`, `while`, ...) ni de références.

1 Exercice : le tri par sélection

L'objectif de cet exercice est d'étudier une implantation particulière d'un algorithme de tri d'une séquence d'entiers.

Déf. III.1 Une séquence s de taille n de valeurs v_i avec $1 \leq i \leq n$ est notée $\langle v_n, \ldots, v_1 \rangle$. Sa taille n est notée $\mid s \mid$. Une même valeur v peut figurer plusieurs fois dans s. Nous noterons $card(v, s)$ le cardinal de v dans s, c'est-à-dire le nombre de fois que v figure dans s, défini par :

$$card(v, \langle v_n, \ldots, v_1 \rangle) = card\{i \in \mathbb{N} \mid 1 \leq i \leq n, v = v_i\}.$$

Le type `entiers` représente une séquence d'entiers. Le type `couple` représente un couple dont le premier élément est un entier et le second est une séquence d'entiers. Les définitions de ces types sont :

```
type entiers == int list;;

type couple == int * entiers;;
```

Soit le programme en langage CaML :

```
let selection s =
  let rec aux c a =
    match a with
      | [] -> (c,a)
      | t::q ->
        if (c < t) then
          let (m,r) = aux c q in
          (m,(t::r))
        else
          let (m,r) = aux t q in
          (m,(c::r))
  in
  aux (hd s) (tl s);;

let rec tri s =
  match s with
    | [] -> []
    | t::q ->
      let (m,r) = selection s in
      m :: (tri r);;
```

Soit la constante *exemple* définie et initialisée par :

```
let exemple = [ 3; 1; 4; 2];;
```

Question III.1 *Détailler les étapes du calcul de* `(trier exemple)` *en précisant pour chaque appel aux fonctions* `selection`, `aux` *et* `trier`, *la valeur du paramètre et du résultat.*

Déf. III.2 (Symbole de Kronecker) Soient deux valeurs v_1 et v_2, le symbole de Kronecker δ est une fonction $\delta(v_1, v_2)$ égale à la valeur 1 si v_1 et v_2 sont égales et à la valeur 0 sinon.

Question III.2 *Soit l'entier* `m`, *soient les séquences d'entiers* `s` $= \langle s_n, \ldots, s_1 \rangle$ *de taille* n *et* `r` $= \langle r_p, \ldots, r_1 \rangle$ *de taille* p, *tels que* `(m, r) = (selection s)`, *montrer que :*

(a) $\forall i, 1 \leq i \leq n, \delta(s_i, \mathtt{m}) + card(s_i, \mathtt{r}) = card(s_i, \mathtt{s})$

(b) $n = p + 1$

(c) $\forall i, 1 \leq i \leq p, \mathtt{m} \leq r_i$

Dans ce but, vous pouvez spécifier les propriétés que doit satisfaire la fonction `aux`. *Montrer que celles-ci sont satisfaites et les exploiter ensuite.*

Question III.3 *Soit la séquence* `s` $= \langle s_m, \ldots, s_1 \rangle$ *de taille* m, *soit la séquence* `r` $= \langle r_n, \ldots, r_1 \rangle$ *de taille* n, *telles que* `r = (trier s)`, *montrer que :*

(a) $m = n$

(b) $\forall i, 1 \leq i \leq m, card(s_i, \mathtt{s}) = card(s_i, \mathtt{r})$

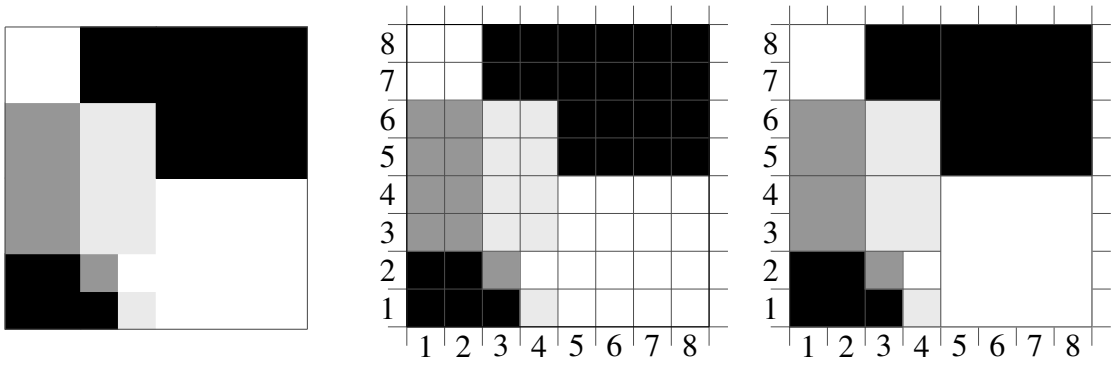
(c) $\forall i, 1 \leq i < m, r_{i+1} \leq r_i$

Question III.4 *Montrer que le calcul des fonctions* `selection`, `aux` *et* `trier` *se termine quelles que soient les valeurs de leurs paramètres respectant le type des fonctions.*

Question III.5 *Donner des exemples de valeurs du paramètre* `s` *de la fonction* `trier` *qui correspondent aux meilleur et pire cas en nombre d'appels récursifs effectués.*
Montrer que la complexité de la fonction `trier` *en fonction du nombre* n *de valeurs dans les séquences données en paramètre est de* $O(\mid n \mid^2)$. *Cette estimation ne prend en compte que le nombre d'appels récursifs effectués.*

2 Problème : représentation d'images par des arbres quaternaires

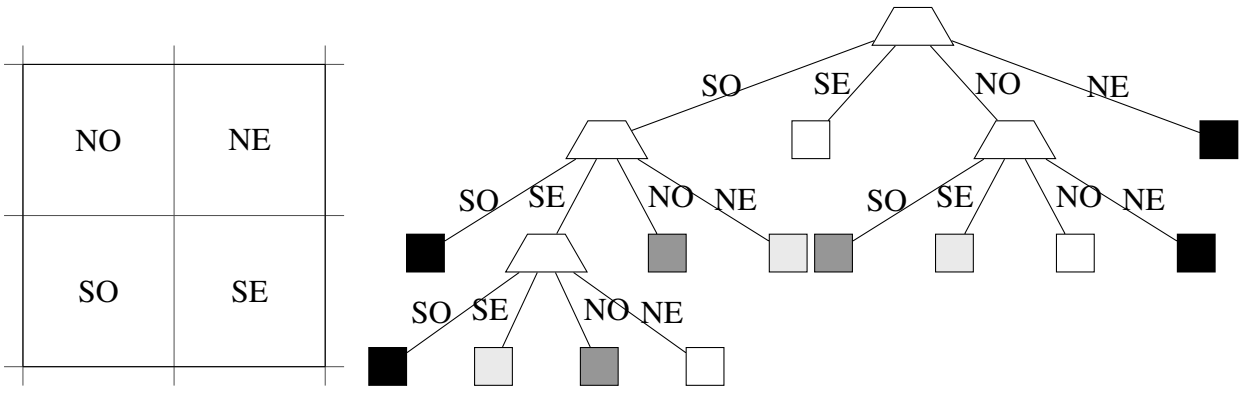
La structure d'arbre quaternaire est une extension de la structure d'arbre binaire qui permet de représenter des informations en deux dimensions. En particulier, la structure d'arbre binaire est utilisée pour représenter de manière plus compacte des images. Il s'agit de décomposer une image par dichotomie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

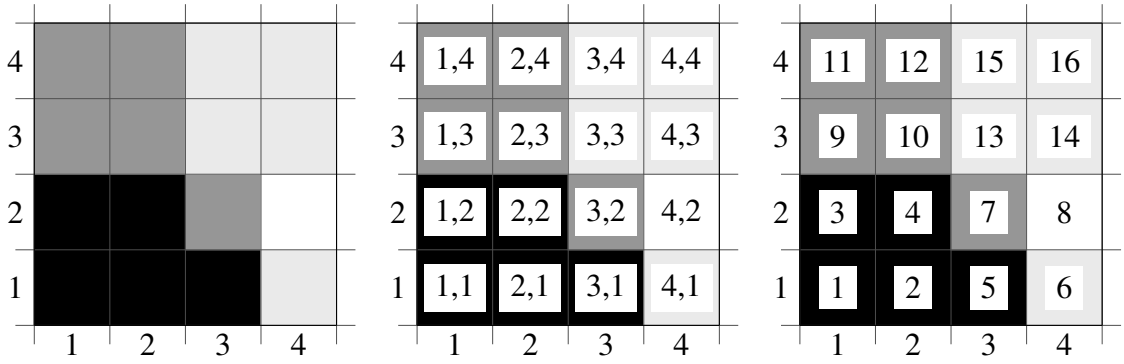
Les images des figures 1 et 2 illustrent ce principe. L'image brute (1-a) contient des carrés de différentes couleurs. Cette image est découpée uniformément en pixels (picture elements) dans l'image (1-b). Les pixels sont des carrés de la plus petite taille nécessaire. Ce découpage fait apparaître de 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.

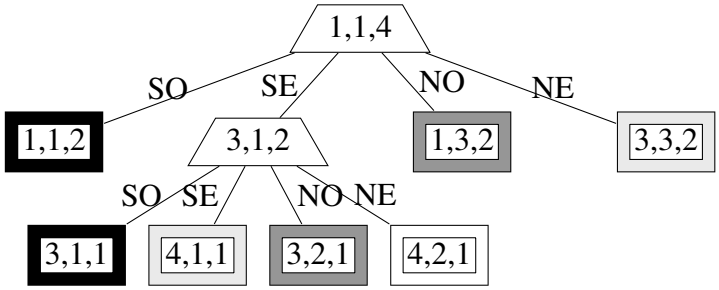


FIGURE 4: structure de données correspondant à l'image (3-a)

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.

Question III.6 *Exprimer le fait qu'un arbre quaternaire a est valide sous la forme d'une propriété* $\mathrm{VAQ}(a)$.

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.*

Partie III : algorithmique et programmation en PASCAL

Cette partie doit être traitée par les étudiants qui ont utilisé le langage PASCAL dans le cadre des enseignements d'informatique. Les fonctions écrites devront être récursives ou faire appel à des fonctions auxiliaires récursives. Elles ne devront pas utiliser d'instructions itératives (c'est-à-dire `for`, `while`, `repeat`, ...).

1 Exercice : le tri par sélection

L'objectif de cet exercice est d'étudier une implantation particulière d'un algorithme de tri d'une séquence d'entiers.

Déf. III.1 Une séquence s de taille n de valeurs v_i avec $1 \leq i \leq n$ est notée $\langle v_n, \ldots, v_1 \rangle$. Sa taille n est notée $\mid s \mid$. Une même valeur v peut figurer plusieurs fois dans s. Nous noterons $card(v, s)$ le cardinal de v dans s, c'est-à-dire le nombre de fois que v figure dans s, défini par :

$$card(v, \langle v_n, \ldots, v_1 \rangle) = card\{i \in \mathbb{N} \mid 1 \leq i \leq n, v = v_i\}.$$

Le type `ENTIERS` représente une séquence d'entiers. Nous supposons prédéfinies les constantes et les fonctions suivantes dont le calcul se termine quelles que soient les valeurs de leurs paramètres. Elles pourront éventuellement être utilisées dans les réponses aux questions :

- `Vide` est une constante de valeur `NIL` qui représente une séquence vide d'entiers ;
- `FUNCTION E_Inserer(e:INTEGER;s:ENTIERS):ENTIERS;` renvoie une séquence d'entiers composée d'un premier entier `e` et du reste de la séquence contenu dans `s` ;
- `FUNCTION E_Tete(s:ENTIERS):INTEGER;` renvoie le premier entier de la séquence `s`. Cette séquence ne doit pas être vide ;
- `FUNCTION E_Queue(s:ENTIERS):ENTIERS;` renvoie le reste de la séquence `s` privée de son premier entier. Cette séquence ne doit pas être vide ;
- `FUNCTION E_Juxtaposer(s1,s2:ENTIERS):ENTIERS;` renvoie une séquence composée des éléments de la séquence `s1` dans le même ordre que dans `s1` puis des éléments de la séquence `s2` dans le même ordre que dans `s2`.

Le type `COUPLE` représente un couple dont le premier élément est un entier et le second est une séquence d'entiers. Nous supposons prédéfinies les constantes et les fonctions suivantes dont le calcul se termine quelles que soient les valeurs de leurs paramètres. Elles pourront éventuellement être utilisées dans les réponses aux questions :

- `FUNCTION C_Creer(v:INTEGER;s:ENTIERS):COUPLE;` renvoie un couple contenant la valeur entière `v` et la séquence d'entiers `s` ;
- `FUNCTION C_Valeur(c:COUPLE):INTEGER;` renvoie la valeur entière située en première partie du couple `c` ;
- `FUNCTION C_Sequence(c:COUPLE):ENTIERS;` renvoie la séquence d'entiers située en seconde partie du couple `c`.

Soit le programme en langage PASCAL :

```
FUNCTION selection(s:ENTIERS):COUPLE;

   FUNCTION aux(c:INTEGER; a:ENTIERS):COUPLE;
   VAR t : INTEGER; q : ENTIERS;
   BEGIN
      IF (a = NIL) THEN
         aux := C_Creer( c, a)
      ELSE BEGIN
         t := E_Tete( a);
         q := E_Queue( a);
         IF (c < t) THEN BEGIN
            i := aux( c, q);
            m := C_Valeur( i);
            r := C_Sequence( i);
            aux := C_Creer( m, E_Ajouter( t, r))
         END ELSE BEGIN
            i := aux( t, q);
            m := C_Valeur( i);
            r := C_Sequence( i);
            aux := C_Creer( m, E_Ajouter( c, r))
         END
      END
   END; (* aux *)

BEGIN
   selection := aux( E_Tete( s), E_Queue( s))
END; (* selection *)

FUNCTION tri(s:ENTIERS):ENTIERS;
VAR i : Couple; m : INTEGER; r : ENTIERS;
BEGIN
   IF (s = NIL) THEN
      tri := s
   ELSE BEGIN
      i := selection( s);
      m := C_Valeur( i);
      r := C_Sequence( i);
      tri := Ajouter( m, tri( r))
   END
END; (* trier *)
```

Soit la constante *exemple* définie et initialisée par :

```
CONST exemple : ENTIERS
   = E_Inserer( 3,  E_Inserer( 1,
                        E_Inserer( 4, E_Inserer(2, Vide))));
```

Question III.1 *Détailler les étapes du calcul de* `trier(exemple)` *en précisant pour chaque appel aux fonctions* `selection`, `aux` *et* `trier`, *la valeur du paramètre et du résultat.*

Déf. III.2 (Symbole de Kronecker) Soient deux valeurs v_1 et v_2, le symbole de Kronecker δ est une fonction $\delta(v_1, v_2)$ égale à la valeur 1 si v_1 et v_2 sont égales et à la valeur 0 sinon.

Question III.2 *Soit l'entier* `m`, *soient les séquences d'entiers* `s` $= \langle s_n, \ldots, s_1 \rangle$ *de taille* n *et* `r` $= \langle r_p, \ldots, r_1 \rangle$ *de taille* p, *soit le couple* `i`, *tels que* `i = selection(s)`, `m = C_Valeur(i)` *et* `r = C_Sequence(i)` *montrer que :*

(a) $\forall i, 1 \leq i \leq n, \delta(s_i, \mathtt{m}) + card(s_i, \mathtt{r}) = card(s_i, \mathtt{s})$

(b) $n = p + 1$

(c) $\forall i, 1 \leq i \leq p, \mathtt{m} \leq r_i$

Dans ce but, vous pouvez spécifier les propriétés que doit satisfaire la fonction `aux`. *Montrer que celles-ci sont satisfaites et les exploiter ensuite.*

Question III.3 *Soit la séquence* `s` $= \langle s_m, \ldots, s_1 \rangle$ *de taille* m, *soit la séquence* `r` $= \langle r_n, \ldots, r_1 \rangle$ *de taille* n, *telles que* `r = trier(s)`, *montrer que :*

(a) $m = n$

(b) $\forall i, 1 \leq i \leq m, card(s_i, \mathtt{s}) = card(s_i, \mathtt{r})$

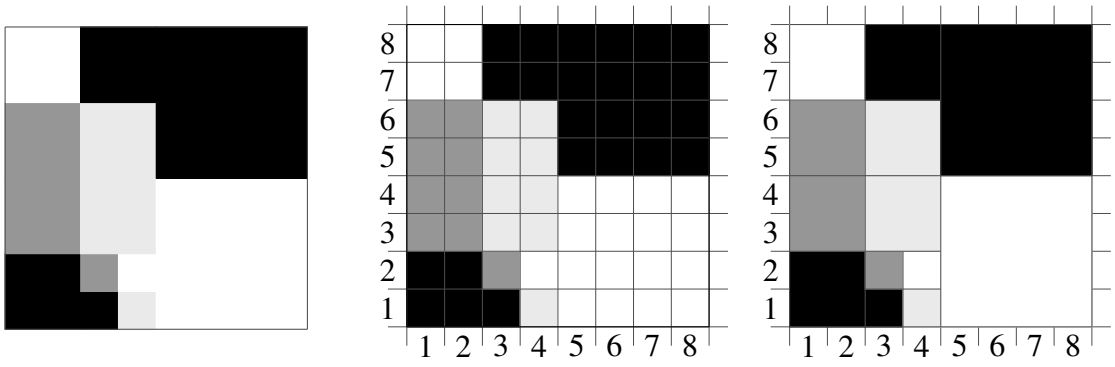
(c) $\forall i, 1 \leq i < m, r_{i+1} \leq r_i$

Question III.4 *Montrer que le calcul des fonctions* `selection`, `aux` *et* `trier` *se termine quelles que soient les valeurs de leurs paramètres respectant le type des fonctions.*

Question III.5 *Donner des exemples de valeurs du paramètre* `s` *de la fonction* `trier` *qui correspondent aux meilleur et pire cas en nombre d'appels récursifs effectués.*
Montrer que la complexité de la fonction `trier` *en fonction du nombre* n *de valeurs dans les séquences données en paramètre est de* $O(\mid n \mid^2)$. *Cette estimation ne prend en compte que le nombre d'appels récursifs effectués.*

2 Problème : représentation d'images par des arbres quaternaires

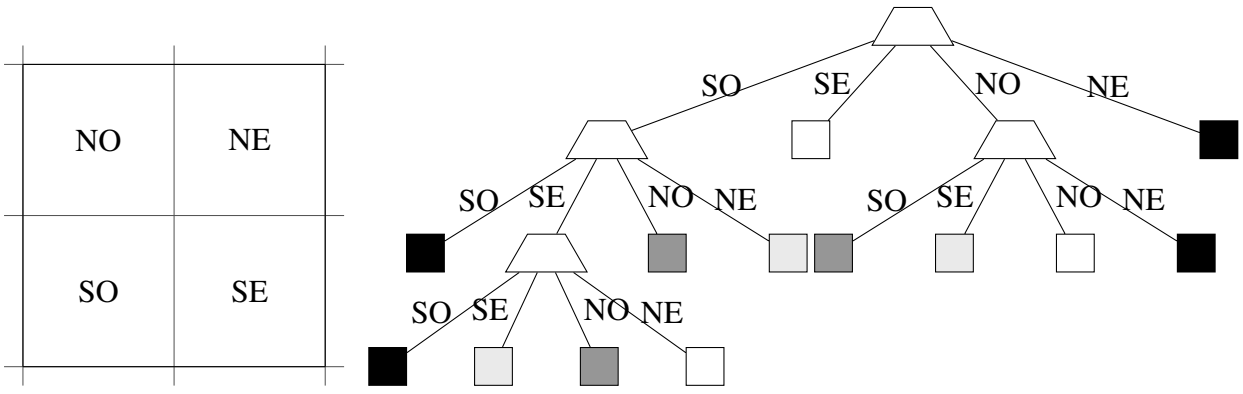
La structure d'arbre quaternaire est une extension de la structure d'arbre binaire qui permet de représenter des informations en deux dimensions. En particulier, la structure d'arbre binaire est utilisée pour représenter de manière plus compacte des images. Il s'agit de décomposer une image par dichotomie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

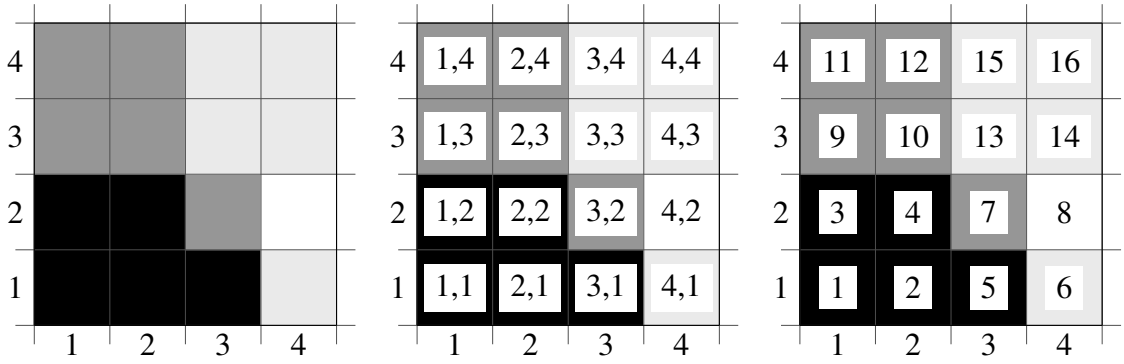
Les images des figures 1 et 2 illustrent ce principe. L'image brute (1-a) contient des carrés de différentes couleurs. Cette image est découpée uniformément en pixels (picture elements) dans l'image (1-b). Les pixels sont des carrés de la plus petite taille nécessaire. Ce découpage fait apparaître de 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.

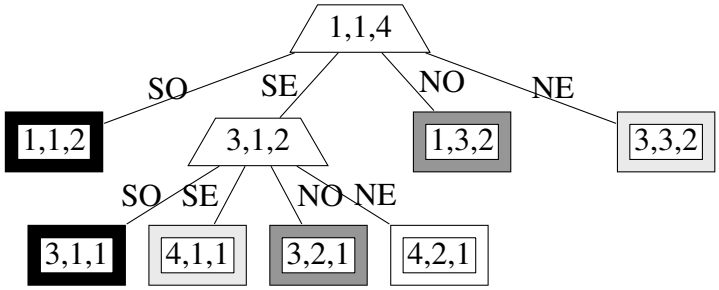


FIGURE 4: structure de données correspondant à l'image (3-a)

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 17 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 21). 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 17) 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.

Question III.6 *Exprimer le fait qu'un arbre quaternaire a est valide sous la forme d'une propriété* $\mathrm{VAQ}(a)$.

2.2 Représentation en PASCAL

Un arbre quaternaire est représenté par le type de base `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).

Nous supposons prédéfinies les constantes et les fonctions suivantes dont le calcul se termine quelles que soient les valeurs de leurs paramètres. Elles pourront éventuellement être utilisées dans les réponses aux questions :

- `FUNCTION Bloc(x,y,t,c:INTEGER):QUATER;` est une fonction qui renvoie un arbre quaternaire dont la racine est un bloc, `x` et `y` sont les coordonnées du point le plus 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 ;
- `FUNCTION Division(x,y,t:INTEGER;so,se,no,ne:QUATER):QUATER;` est une fonction qui renvoie un arbre quaternaire dont la racine est une division, `x` et `y` sont les coordonnées du point le plus 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) ;
- `FUNCTION EstBloc(a:QUATER):BOOLEAN;` est une fonction qui renvoie la valeur `TRUE` si la racine de l'arbre `a` est un bloc et la valeur `FALSE` sinon ;
- `FUNCTION Abscisse(a:QUATER):QUATER;` est une fonction qui, appliquée sur un arbre quaternaire `a`, renvoie l'abscisse du point situé le plus en bas à gauche de l'image représentée par `a` ;
- `FUNCTION Ordonnée(a:QUATER):QUATER;` est une fonction qui, appliquée sur un arbre quaternaire `a`, renvoie l'ordonnée du point situé le plus en bas à gauche de l'image représentée par `a` ;
- `FUNCTION Taille(a:QUATER):QUATER;` est une fonction qui, appliquée sur un arbre quaternaire `a`, renvoie la longueur du côté de l'image représentée par `a` ;
- `FUNCTION Couleur(a:QUATER):QUATER;` est une fonction qui, appliquée sur un arbre quaternaire `a` dont la racine est un bloc, renvoie la couleur de l'image représentée par `a` ;
- `FUNCTION Fils(p:POSITION;a:QUATER):QUATER;` est une fonction qui, appliquée sur un arbre quaternaire `a` dont la racine est une division, renvoie le fils de `a` situé à la position `p` dans cette division.

Exemple III.3 En associant les valeurs `0`, `1`, `2`, `3` aux couleurs noir, gris foncé, gris clair et blanc, le programme suivant :

```
VAR
  exemple : QUATER;
  B11_2, B31_1 B41_1 B32_1 B42_1 D31_2 B13_2 B33_2 : QUATER;
CONST
  B11_2 := Bloc( 1, 1, 2, 0); (* SO racine *)
  B31_1 := Bloc( 3, 1, 1, 0); (* SO du SE racine  *)
  B41_1 := Bloc( 4, 1, 1, 2); (* SE du SE racine  *)
  B32_1 := Bloc( 3, 2, 1, 1); (* NO du SE racine  *)
  B42_1 := Bloc( 4, 2, 1, 3); (* NE du SE racine  *)
  D31_2 := (* SE racine *)
    Division( 3, 1, 2, B31_1, B41_1, B32_1, B42_1);
  B13_2 := Bloc( 1, 3, 2, 1); (* NO racine *)
  B33_2 := Bloc( 3, 3, 2, 2); (* NE racine *)
BEGIN
  exemple := (* racine *)
    Division( 1, 1, 4, B11_2, B31_2, B13_2, B33_2)
END
```

affecte à la variable *exemple* l'arbre quaternaire représenté graphiquement dans la figure 4 (page 18).

2.2.1 Scission d'un arbre quaternaire

Question III.7 *Ecrire en PASCAL une fonction* `scinder(a: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 PASCAL une fonction* `fusionner(so,se,no,ne: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 PASCAL une fonction* `profondeur(a:QUATER):INTEGER;` *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 PASCAL une fonction* `consulter(x,y:INTEGER;a:QUATER):INTEGER;` *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 PASCAL une fonction* `peindre(x,y,c:INTEGER;a: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 PASCAL une fonction* `valider(a:QUATER):BOOLEAN;` *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 18) 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 PASCAL

Une séquence de positions et de couleurs est représentée par le type *`SEQUENCE`* et le type *`POINT`* qui contient la position et la couleur associée. Nous supposons prédéfinies les constantes et les fonctions suivantes dont le calcul se termine quelles que soient les valeurs de leurs paramètres. Elles pourront éventuellement être utilisées dans les réponses aux questions :

- *`Vide`* est une constante de valeur *`NIL`* qui représente une séquence vide ;
- *`FUNCTION P_Creer(p,c:INTEGER):POINT;`* renvoie un point contenant la position *`p`* et la couleur *`c`* ;
- *`FUNCTION P_Position(p:POINT):INTEGER;`* renvoie la position contenue dans le point *`p`* ;

— *FUNCTION P_Couleur(p:POINT):INTEGER;* renvoie la couleur contenue dans le point *p* ;
— *FUNCTION S_Inserer(p:POINT;s:SEQUENCE):SEQUENCE;* renvoie une séquence de points composée d'un premier point *p* et du reste de la séquence contenu dans *s* ;
— *FUNCTION S_Tete(s:SEQUENCE):POINT;* renvoie le premier point de la séquence *s*. Cette séquence ne doit pas être vide ;
— *FUNCTION S_Queue(s:SEQUENCE):SEQUENCE;* renvoie le reste de la séquence *s* privée du premier point. Cette séquence ne doit pas être vide ;
— *FUNCTION S_Juxtaposer(s1,s2:SEQUENCE):SEQUENCE;* renvoie une séquence composée des éléments de la séquence *s1* dans le même ordre que dans *s1* puis des éléments de la séquence *s2* dans le même ordre que dans *s2*.

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 18) produit la séquence calculée par :

```
S_Inserer( P_Creer(1,0), S_Inserer( P_Creer(2,0),
S_Inserer( P_Creer(3,0), S_Inserer( P_Creer(4,0),
S_Inserer( P_Creer(5,0), S_Inserer( P_Creer(6,2),
S_Inserer( P_Creer(7,1), S_Inserer( P_Creer(8,3),
S_Inserer( P_Creer(9,1), S_Inserer( P_Creer(10,1),
S_Inserer( P_Creer(11,1), S_Inserer( P_Creer(12,1),
S_Inserer( P_Creer(13,2), S_Inserer( P_Creer(14,2),
S_Inserer( P_Creer(15,2), S_Inserer( P_Creer(16,2),
Vide))))))))))))))))
```

2.3.2 Sauvegarde d'un arbre quaternaire

Question III.13 *Ecrire en PASCAL une fonction* `sauvegarder(a: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 PASCAL une fonction* `restaurer(s: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.*

Fin de l'énoncé

Travailler les chapitres de ce sujet

Prepa Help suit le programme de MP chapitre par chapitre : cours détaillé, méthodes, pièges, exercices et problèmes corrigés pas à pas, tirés des concours. L'essai gratuit donne accès aux chapitres offerts de ta filière, sans carte bancaire.

Chapitres liés à ce sujet

- Logique, ensembles et applications
- Langage OCaml : expressions, types, filtrage, récursivité
- Spécifier, tester, prouver : terminaison et correction
- Tris par sélection, à bulles, par insertion et par comptage
- Images : matrices de pixels, opérations, convolution
- Récursivité



Ouvrir l'essai gratuit

Scanne le code ou clique sur le bouton.

prepahelp.com/#essai

Une question sur le concours ou sur ta filière :
écrire sur WhatsApp

Les autres années et épreuves : prepahelp.com/annales/ccinp/mp