SESSION 2009

CONCOURS COMMUNS POLYTECHNIQUES

MPIN007

EPREUVE SPECIFIQUE - FILIERE MP

INFORMATIQUE

Durée : 3 heures

Les calculatrices sont interdites.

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.

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

Partie I : Logique et calcul des propositions

Vous êtes l'ambassadeur de la fédération des planètes unies auprès d'une civilisation extra-terrestre particulièrement susceptible. Il est extrêmement important de respecter leur protocole en ce qui concerne la couleur des vêtements. Pour éviter les incompréhensions, vous disposez d'un assistant de la dernière génération d'intelligence artificielle bio-électronique qui contient toutes les informations disponibles sur leur culture. Mais celui-ci semble perturbé depuis votre arrivée. Il vous donne maintenant des indications sous la forme d'énigmes. Vous avez constaté qu'il vous communiquait systématiquement trois affirmations dont exactement l'une des trois est correcte et les deux autres fausses.

Nous noterons A_1, A_2 et A_3 les propositions associées aux trois affirmations communiquées par votre assistant.

Question I.1 *Représenter cette règle sous la forme d'une formule du calcul des propositions dépendant de A_1, A_2 et A_3.*

Vous devez d'abord rencontrer le gouverneur de la capitale qui est le seul à pouvoir vous obtenir un contact avec le conseil des sages.

Votre assistant vous donne les indications suivantes :
- Porte au moins un vêtement de couleur claire, mais aucun vêtement de couleur sombre !
- Si tu portes au moins un vêtement de couleur claire alors ne porte aucun vêtement de couleur sombre !
- Ne porte aucun vêtement de couleur sombre !

Nous noterons C, respectivement S, les variables propositionnelles correspondant au fait de porter au moins un vêtement de couleur claire, respectivement sombre.

Question I.2 *Exprimer A_1, A_2 et A_3 sous la forme de formules du calcul des propositions dépendant de C et de S.*

Question I.3 *En utilisant le calcul des propositions (résolution avec les formules de De Morgan), déterminer quelle(s) couleur(s) de vêtement vous devez porter pour rencontrer le gouverneur.*

Grâce à ces conseils avisés, vous obtenez un rendez-vous avec le conseil. Vous disposez de vêtements de couleur rouge, verte et bleue. Votre assistant vous transmet ses conseils mais la communication est brouillée juste à la fin et vous ne pouvez plus le contacter avant l'heure du rendez-vous. Voici ce que vous avez réussi à comprendre :
- Ne porte un vêtement de couleur rouge que si tu portes aussi un vêtement de couleur bleue !
- Si tu ne portes pas de vêtement de couleur verte alors ne porte pas de vêtement de couleur bleue !
- Porte au moins un vêtement de couleur bleue ou de couleur ... !

Nous noterons R, V et B les variables propositionnelles correspondant au fait de porter au moins un vêtement de couleur rouge, verte et bleue.

Question I.4 *Exprimer A_1, A_2 et A_3 sous la forme de formules du calcul des propositions dépendant de R, V et B.*

Question I.5 *En utilisant le calcul des propositions (résolution avec les tables de vérité), déterminer quelle(s) couleur(s) de vêtement vous devez porter pour votre entrevue avec le conseil des sages.*

Partie II : 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 (`for`, `while`, ...) ni de références.

La gestion de la mémoire est une fonction essentielle des systèmes d'exploitation des ordinateurs. Il s'agit de permettre, d'une part l'allocation de morceaux, aussi appelés blocs, de mémoire disponible à la demande des programmes utilisateurs, et d'autre part la libération des morceaux de mémoire qui ne sont plus utilisés.

Nous appellerons blocs alloués, respectivement blocs libres, les morceaux de mémoire qui sont, respectivement ne sont pas, utilisés.

L'objectif de ce problème est d'étudier une réalisation particulière pour la gestion de la mémoire exploitant une structure d'arbre binaire pour représenter les blocs. Cette structure permet de réduire la complexité en temps de calcul des opérations d'allocation et de libération des blocs.

1 Préliminaires : Représentation des blocs

1.1 Définitions

Déf. II.1 (Mémoire) *La mémoire est représentée comme un intervalle de* $\mathbb{N}$.

Déf. II.2 (Bloc de mémoire) *Un bloc de mémoire est un sous-intervalle de la mémoire qui sera caractérisé par son adresse et sa taille. L'adresse d'un bloc est aussi appelée sa base. Soit* b *un bloc, nous noterons* $\mathcal{T}(b)$ *sa taille et* $\mathcal{B}(b)$ *sa base. L'intervalle correspondant dans la mémoire est :*

$$[\mathcal{B}(b), \mathcal{B}(b) + \mathcal{T}(b)[= [\mathcal{B}(b), \mathcal{B}(b) + \mathcal{T}(b) - 1]$$

Déf. II.3 (Bloc libre indivisible) *Un bloc libre est indivisible s'il ne peut pas être décomposé en blocs plus petits lors d'une allocation.*

Déf. II.4 (Blocs adjacents) *Deux blocs sont adjacents quand la fin d'un bloc se situe exactement avant le début de l'autre bloc, c'est-à-dire quand la base du second bloc est égale à la base du premier augmentée de sa taille.*

Déf. II.5 (Fusion de blocs) *La fusion de deux blocs adjacents a comme base la plus petite des bases des deux blocs, et comme taille la somme des tailles des deux blocs. Cette opération correspond à l'union des intervalles correspondant à chaque bloc.*

Déf. II.6 (Blocs disjoints) *Deux blocs sont disjoints si l'intersection des intervalles correspondants est vide.*

1.2 Représentation en CaML

Nous ferons l'hypothèse que les bases, respectivement les tailles, manipulées sont positives, respectivement strictement positives.

Un bloc de mémoire est représenté par le type `bloc` équivalent à une paire de `int` (sa base et sa taille).

```
type bloc == int * int;;
```

Nous allons maintenant définir plusieurs opérations sur les blocs.

1.3 Opérations sur la structure de bloc

La première opération est le test si deux blocs sont adjacents.

Question II.1 *Écrire en CaML une fonction* `adjacent` *de type* `bloc -> bloc -> bool` *telle que l'appel* `(adjacent b1 b2)` *renvoie la valeur* `true` *si le bloc* `b1` *est adjacent au bloc* `b2`*, et la valeur* `false` *sinon.*

La seconde opération est la fusion de deux blocs.

Question II.2 *Écrire en CaML une fonction* `fusion` *de type* `bloc -> bloc -> bloc` *telle que l'appel* `(fusion b1 b2)` *renvoie un bloc correspondant à la fusion de* `b1` *et* `b2` *si* `b1` *est adjacent au bloc* `b2`*, et renvoie le bloc de base* `-1` *et de taille* `-1` *sinon.*

2 Préliminaires : Codage binaire des nombres entiers positifs

Le codage binaire de taille t d'un nombre entier positif strictement inférieur à 2^t est représenté par la séquence $\langle b_t, \ldots, b_1 \rangle$ de chiffres binaires, ou bits (0 ou 1).

Notons que le premier chiffre est le bit de poids le plus fort et le dernier chiffre le bit de poids le plus faible. Ceci est essentiel pour la question II.11.

2.1 Définitions

Déf. II.7 (Codage binaire d'un nombre entier positif) *Soit un nombre entier positif* $n \in \mathbb{N}$*, le codage binaire unique de taille* t *du nombre entier positif* n *(avec* $n < 2^t$*) est la séquence* $\langle b_t, \ldots, b_1 \rangle$ *telle que :*

$$\forall i, 1 \leq i \leq t \Rightarrow b_i \in \{0, 1\}$$
$$n = \sum_{i=1}^{t} b_i \times 2^{i-1}$$

Question II.3 *Calculer la valeur de l'indice du chiffre binaire de valeur* 1 *de poids le plus fort du codage binaire du nombre entier positif* n *en fonction de la valeur de* n.

Question II.4 *Montrer que tout nombre entier strictement positif s'écrit de manière unique sous la forme du produit d'un nombre entier positif impair et d'une puissance de* 2.

2.2 Représentation en CaML

Le codage binaire d'un nombre entier positif est représenté par le type `binaire` équivalent à une liste de `int`.

```
type binaire == int list;;
```

Son parcours sera donc effectué de la même manière que celui d'une liste.

2.3 Nombre de bits nécessaire pour un codage

La première opération réalisée est le calcul du nombre de bits nécessaire pour le codage d'un nombre entier positif au format binaire.

Question II.5 *Écrire en CaML une fonction `nombre` de type `int -> int` telle que l'appel `(nombre e)` renvoie le nombre de chiffres binaires nécessaires pour coder le nombre entier positif `e` au format binaire. Cette fonction pourra être récursive ou faire appel à des fonctions auxiliaires récursives.*

2.4 Codage d'un nombre entier positif au format binaire

La seconde opération réalisée est le codage d'un nombre entier positif au format binaire.

Question II.6 *Écrire en CaML une fonction `codage` de type `int -> binaire` telle que l'appel `(codage n)` renvoie une liste d'entiers composée des coefficients $b_t, \ldots, b_1$ correspondant au codage binaire de `n` en utilisant le nombre de bits nécessaire pour `n`. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

3 Réalisation à base d'arbres binaires

L'utilisation de la structure d'arbre binaire pour représenter les blocs de mémoire permet de réduire la complexité en temps de calcul pour les opérations d'allocation et de libération.

L'ensemble de la mémoire, blocs libres et blocs alloués, est représentée par un arbre binaire dont les feuilles sont les blocs et les nœuds correspondent à une décomposition en deux parties égales de la mémoire.

Les blocs libres indivisibles sont donc de taille impaire.

3.1 Arbre binaire étiqueté

3.1.1 Définition

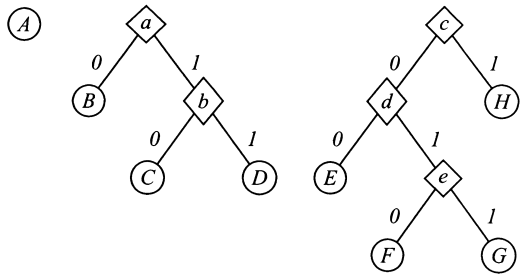
Déf. II.8 (Arbre binaire étiqueté) *Un arbre binaire étiqueté a est une structure qui peut :*

- *soit être une feuille qui peut contenir une étiquette,*
- *soit un nœud qui contient un sous-arbre gauche (noté $\mathcal{G}(a)$) et un sous-arbre droit (noté $\mathcal{D}(a)$) qui sont tous deux des arbres binaires étiquetés, et peut également contenir une étiquette.*

L'ensemble des feuilles, respectivement des nœuds, est noté $\mathcal{F}$, respectivement $\mathcal{N}$. L'ensemble des arbres est $\mathcal{A} = \mathcal{F} \cup \mathcal{N}$.

L'ensemble des feuilles, respectivement des nœuds, d'un arbre a est noté $\mathcal{F}(a)$, respectivement $\mathcal{N}(a)$.

Exemple II.1 (Arbres binaires étiquetés) *Voici trois exemples d'arbres binaires dont les nœuds, respectivement les feuilles, sont étiquetés par des minuscules, respectivement des majuscules, (les nœuds sont des losanges, les feuilles sont des cercles). Les arcs sont étiquetés par des chiffres binaires (0 pour le fils gauche et 1 pour le fils droit) qui seront utilisés pour représenter les chemins qui conduisent de la racine aux nœuds et aux feuilles dans l'arbre (voir la définition II.12) :*



3.1.2 Parcours gauche-droite d'un arbre binaire

Déf. II.9 (Parcours gauche-droite d'un arbre binaire) *Le parcours gauche-droite d'un arbre binaire a, noté $\mathcal{E}(a)$, construit la séquence des feuilles de l'arbre dans l'ordre de gauche à droite. Il est défini par les équations :*

$$\mathcal{E}(a) = \begin{cases} \langle a \rangle & \text{si } a \in \mathcal{F} \\ \langle f_1, \ldots, f_{m+n} \rangle & \text{si } a \in \mathcal{N} \\ & \text{et } \langle f_1, \ldots, f_m \rangle = \mathcal{E}(\mathcal{G}(a)) \\ & \text{et } \langle f_{m+1}, \ldots, f_{m+n} \rangle = \mathcal{E}(\mathcal{D}(a)) \end{cases}$$

Exemple II.2 (Parcours gauche-droite) *Les résultats du parcours gauche-droite des trois arbres de l'exemple II.1 sont respectivement $\langle A \rangle$, $\langle B, C, D \rangle$ et $\langle E, F, G, H \rangle$.*

3.1.3 Profondeur dans un arbre

Déf. II.10 (Profondeur dans un arbre) *La profondeur d'un élément e (nœud ou feuille) dans un arbre a est égale au nombre d'arcs entre la racine et cet élément dans l'arbre. Nous la noterons $\mathcal{P}(e, a)$.*

Exemple II.3 (Profondeur dans un arbre) *Les profondeurs des éléments A, b et e dans les trois arbres de l'exemple II.1 sont respectivement 0, 1 et 2.*

Déf. II.11 (Profondeur d'un arbre) *La profondeur d'un arbre a est égale au maximum de la profondeur de toutes ses feuilles. Nous la noterons $\mathcal{P}(a)$.*

Exemple II.4 (Profondeur d'un arbre) *Les profondeurs des arbres de l'exemple II.1 sont respectivement 0, 2 et 3.*

3.1.4 Chemin dans un arbre

Déf. II.12 (Chemin dans un arbre) *Le chemin d'un élément e (nœud ou feuille) dans un arbre a est la séquence des étiquettes des arcs entre la racine et cet élément dans l'arbre. Nous le noterons $\mathcal{C}(e, a)$.*

Notons que le chemin de la racine d'un arbre est vide.

Notons qu'un chemin correspond au codage binaire d'un nombre entier positif dont le bit de poids le plus fort est l'étiquette la plus proche de la racine.

Exemple II.5 (Chemin d'un élément) *Les chemins des éléments A, C et G dans l'exemple II.1 sont respectivement $\langle\rangle$, $\langle 1, 0\rangle$ et $\langle 0, 1, 1\rangle$.*

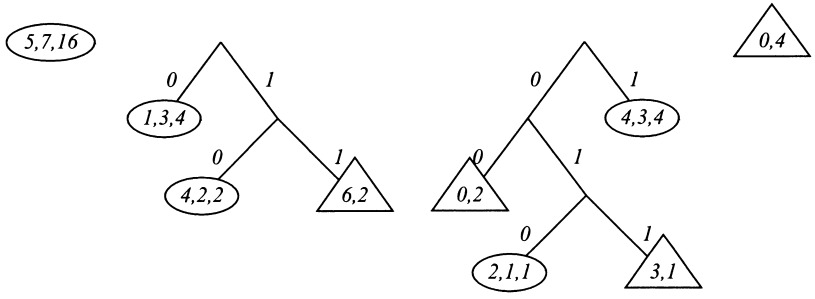
3.2 Arbres binaires de blocs

Nous considérerons par la suite des arbres binaires dont les feuilles sont de deux types, celles qui représentent un bloc libre, et celles qui représentent un bloc alloué. Toutes les feuilles sont décorées par la base et la taille du bloc. Les feuilles allouées sont également décorées par la taille utilisée dans le bloc.

Déf. II.13 (Arbre binaire de blocs) *Un arbre binaire de blocs est un arbre binaire étiqueté dont :*

- *Chaque feuille f est :*
  - *soit une feuille libre qui contient deux étiquettes entières, sa base (notée $\mathcal{B}(f)$) et sa taille (notée $\mathcal{T}(f)$), l'ensemble des feuilles libres est noté $\mathcal{F}_L$,*
  - *soit une feuille occupée qui contient trois étiquettes entières, sa base (notée $\mathcal{B}(f)$), sa taille allouée (notée $\mathcal{O}(f)$) et sa taille totale (notée $\mathcal{T}(f)$), l'ensemble des feuilles occupées est noté $\mathcal{F}_O$ ($\mathcal{F} = \mathcal{F}_L \cup \mathcal{F}_O$).*
- *Les nœuds ne sont pas étiquetés.*
- *Les fils gauche et droit de chaque nœud sont des arbres binaires de blocs.*

Exemple II.6 (Arbres binaires de blocs) *Voici quatre exemples d'arbres binaires de blocs (les feuilles libres sont des triangles contenant la base puis la taille, les feuilles occupées sont des ovales contenant la base, puis la taille occupée puis la taille totale). Les arcs sont étiquetés par des chiffres binaires (0 pour le fils gauche et 1 pour le fils droit) qui seront utilisés pour représenter les chemins dans l'arbre (voir la définition II.12) :*



Déf. II.14 (Taille d'un arbre de blocs) *La taille totale, notée $\mathcal{T}(a)$, d'un arbre de blocs a est la somme de la taille de tous les blocs qui composent ses feuilles.*

Exemple II.7 (Taille d'un arbre de blocs) *Les tailles des arbres de l'exemple II.6 sont respectivement 16, 8, 8 et 4.*

Déf. II.15 (Taille maximum disponible) *La taille maximum disponible, notée $\mathcal{M}(a)$, d'un arbre de blocs a est la taille du plus grand des blocs libres qui composent ses feuilles.*

Exemple II.8 (Taille maximum disponible) *Les tailles maximum disponibles des arbres de l'exemple II.6 sont respectivement 0, 2, 2 et 4.*

Déf. II.16 (Base d'un arbre de blocs) *La base, notée $\mathcal{B}(a)$, d'un arbre de blocs a est la base du bloc contenu dans la feuille la plus à gauche de l'arbre.*

Exemple II.9 (Base d'un arbre de blocs) *Les bases des arbres de l'exemple II.6 sont respectivement 5, 1, 0 et 0.*

Déf. II.17 (Arbre d'allocation) *Un arbre d'allocation est un arbre de blocs qui vérifie les contraintes suivantes :*

C1 *Les fils gauche et droit d'un même nœud ne sont pas simultanément des feuilles libres.*

C2 *Les fils gauche et droit d'un même nœud sont de même taille.*

C3 *La base du fils droit d'un nœud est égale à la base de ce nœud augmentée de la taille du fils gauche.*

C4 *La taille allouée dans une feuille occupée est supérieure à la moitié de la taille totale de la feuille.*

Exemple II.10 (Arbres d'allocation) *Le premier et le deuxième arbres de l'exemple II.6 ne sont pas des arbres d'allocation car ils ne respectent pas les contraintes **C4** (le premier) et **C3** (le deuxième). Le troisième et le quatrième arbres de l'exemple II.6 sont des arbres d'allocation.*

Question II.7 *Exprimer la définition II.17 sous la forme d'une propriété $AA(e)$ qui indique que a est un arbre d'allocation. Vous pouvez utiliser pour cela les fonctions définies sur les arbres binaires de blocs.*

Déf. II.18 (Arbre de blocs dense) *Un arbre de blocs est dense si et seulement si chaque feuille dans un parcours gauche-droite est adjacente à sa voisine de droite.*

Question II.8 *Montrer qu'un arbre d'allocation est un arbre de blocs dense.*

Question II.9 *Soit a un arbre d'allocation, soit a', un sous-arbre de a, montrer que :* $\mathcal{T}(a) = \mathcal{T}(a') \times 2^{\mathcal{P}(a',a)}$.

Question II.10 *Montrer que les blocs libres indivisibles dans un arbre d'allocation doivent tous être de la même taille et de la même profondeur.*

Question II.11 *Montrer que la base d'un bloc b, distinct de a, dans un arbre d'allocation a est égale à la base de a augmentée du nombre entier positif dont le codage binaire est le chemin de b dans a que multiplie la taille de b.*

3.2.1 Représentation des arbres binaires de blocs en CaML

Pour simplifier le problème, nous ne considérerons que des mémoires dont la taille est une puissance de 2 (les blocs libres indivisibles seront donc de taille 1) et qui commencent à la base 0 donc des intervalles de la forme $[0, 2^n - 1]$ avec $n \in \mathbb{N}$.

Pour simplifier le problème, nous n'étiquetons pas les blocs avec leurs bases mais uniquement leurs tailles. Nous pouvons utiliser le chemin et la taille pour calculer la base d'un bloc en s'appuyant sur le résultat de la question II.11. Nous utiliserons donc par la suite le chemin au lieu de la base pour simplifier les fonctions.

Un arbre binaire de blocs est représenté par le type CaML :

```
type arbre = Libre of int
           | Occupe of int * int
           | Noeud of arbre * arbre;;
```

Dans l'appel `Libre( t)`, le paramètre `t` est la taille de l'espace libre.

Dans l'appel `Occupe( to, tt)`, les paramètres `to` et `tt` sont respectivement la taille de l'espace occupé dans le bloc et la taille totale du bloc.

Dans l'appel `Noeud( fg, fd)`, les paramètres `fg` et `fd` sont respectivement le fils gauche et le fils droit de la racine de l'arbre créé.

Exemple II.11 *Le terme suivant*

```
Noeud(
   Noeud(
      Libre( 2 ),
      Noeud(
         Occupe( 1, 1),
         Libre( 1)
      )
   ),
   Occupe( 3, 4)
)
```

est alors associé au troisième arbre binaire de blocs représenté graphiquement dans l'exemple II.6.

3.3 Opérations sur la structure d'arbre d'allocation

3.3.1 Espace libre maximum d'un arbre d'allocation

La première opération est le calcul de l'espace libre maximum disponible dans un arbre d'allocation.

Question II.12 *Écrire en CaML une fonction* `maximum` *de type* `arbre -> int` *telle que l'appel* `(maximum a)` *renvoie la taille du plus grand bloc libre de l'arbre d'allocation* `a`*, c'est-à-dire* $\mathcal{M}(\mathtt{a})$*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

3.3.2 Vérification de la structure d'ensemble d'allocation

La deuxième opération est la vérification qu'un arbre de blocs est un arbre d'allocation, c'est-à-dire respecte les contraintes de la définition II.17.

Question II.13 *Écrire en CaML une fonction* `verifier` *de type* `arbre -> bool` *telle que l'appel* `(verifier a)` *sur un arbre d'allocation* `a` *renvoie la valeur* `true` *si l'arbre* `a` *respecte les contraintes d'un arbre d'allocation, et la valeur* `false` *sinon. L'algorithme utilisé ne devra parcourir qu'une seule fois l'arbre. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

3.3.3 Libération d'un bloc

La troisième opération est la libération d'un bloc dans un arbre d'allocation.

Question II.14 *Construire l'arbre d'allocation qui résulte de la libération du bloc dont la base est* 2 *dans le troisième arbre d'allocation de l'exemple II.6. Préciser les différentes étapes qui permettent de construire cet arbre.*

Question II.15 *Écrire en CaML une fonction* `liberer` *de type* `binaire -> arbre -> arbre` *telle que l'appel* `(liberer c a)` *sur un arbre d'allocation* `a` *qui contient le bloc alloué de chemin* `c` *renvoie un arbre d'allocation contenant :*

- *les blocs alloués de* `a` *sauf le bloc de chemin* `c`*,*
- *le résultat de la fusion des blocs libres de* `a` *et du bloc de chemin* `c` *pour que le résultat respecte la structure d'arbre d'allocation.*
- *les blocs libres de* `a` *qui ne sont pas adjacents au bloc de chemin* `c`*,*
- *le bloc libéré de chemin* `c` *si aucun bloc libre de* `a` *ne lui est adjacent.*

Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.

Question II.16 *Donner des exemples de valeurs des paramètres* `c` *et* `a` *de la fonction* `liberer` *qui correspondent aux meilleur et pire cas en nombre d'appels récursifs effectués.*

Calculer une estimation de la complexité dans les meilleur et pire cas de la fonction `liberer` *en fonction de la profondeur de l'arbre d'allocation* `a`*. Cette estimation ne prendra en compte que le nombre d'appels récursifs effectués.*

3.3.4 Allocation d'un bloc

La quatrième opération est l'allocation d'un bloc dans un arbre d'allocation.

De nombreuses stratégies sont possibles pour choisir le bloc libre dans l'arbre qui sera utilisé parmi ceux dont la taille est supérieure ou égale à la taille demandée. Nous allons appliquer la stratégie consistant à choisir le bloc le plus petit parmi ceux-ci.

Lorsque le bloc choisi est de taille strictement supérieure à la taille demandée, ce bloc doit être décomposé en un sous-arbre qui contiendra le bloc alloué et un ou plusieurs blocs libres pour préserver la structure d'arbre d'allocation.

Question II.17 *Donner le chemin et construire l'arbre d'allocation qui résultent de l'allocation de la taille* 1 *dans le quatrième arbre d'allocation de l'exemple II.6. Donner les différentes étapes qui permettent de construire ce chemin et cet arbre.*

Question II.18 *Écrire en CaML une fonction* `allouer` *de type* `int -> arbre -> binaire * arbre` *telle que l'appel* `(allouer t a)` *sur un arbre d'allocation* `a` *renvoie une paire composée d'un chemin vers un bloc et d'un arbre d'allocation.*

- *Si* `a` *ne contient pas de bloc libre de taille supérieure ou égale à* `t`*, alors le chemin renvoyé contient uniquement le nombre entier* `-1`*, et l'arbre d'allocation renvoyé contient exactement les mêmes blocs que* `a`*.*
- *Si* `a` *contient au moins un bloc de taille supérieure ou égale à* `t` *alors, notons* `b` *un bloc parmi les plus petits blocs de* `a` *de taille supérieure ou égale à* `t`*,*
  - *le chemin renvoyé est celui d'un bloc alloué qui aura la même base que* `b` *et la taille* `t`*,*
  - *l'arbre d'allocation renvoyé contient :*
    - *les mêmes blocs que* `a` *sauf* `b`*,*
    - *les blocs qui résultent de la décomposition de* `b` *pour réaliser l'allocation en respectant les contraintes d'un arbre d'allocation, dont le bloc alloué.*

Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.

Question II.19 *Donner des exemples de valeurs des paramètres* `t` *et* `a` *de la fonction* `allouer` *qui correspondent aux meilleur et pire cas en nombre d'appels récursifs effectués.*

Calculer une estimation de la complexité dans les meilleur et pire cas de la fonction `allouer` *en fonction de la taille de l'arbre d'allocation* `a`*. Cette estimation ne prendra en compte que le nombre d'appels récursifs effectués.*

Question II.20 *Comparer les estimations des complexités en nombre d'appels récursifs dans les meilleur et pire cas pour la fonction* `allouer` *pour la structure d'ensemble d'allocation et la structure d'arbre d'allocation en fonction de la taille de la mémoire.*

Partie II : 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 (`for`, `while`, `repeat`, ...).

La gestion de la mémoire est une fonction essentielle des systèmes d'exploitation des ordinateurs. Il s'agit de permettre, d'une part l'allocation de morceaux, aussi appelés blocs, de mémoire disponible à la demande des programmes utilisateurs, et d'autre part la libération des morceaux de mémoire qui ne sont plus utilisés.

Nous appellerons blocs alloués, respectivement blocs libres, les morceaux de mémoire qui sont, respectivement ne sont pas, utilisés.

L'objectif de ce problème est d'étudier une réalisation particulière pour la gestion de la mémoire exploitant une structure d'arbre binaire pour représenter les blocs. Cette structure permet de réduire la complexité en temps de calcul des opérations d'allocation et de libération des blocs.

1 Préliminaires : Représentation des blocs

1.1 Définitions

Déf. II.1 (Mémoire) *La mémoire est représentée comme un intervalle de* $\mathbb{N}$.

Déf. II.2 (Bloc de mémoire) *Un bloc de mémoire est un sous-intervalle de la mémoire qui sera caractérisé par son adresse et sa taille. L'adresse d'un bloc est aussi appelée sa base. Soit* b *un bloc, nous noterons* $\mathcal{T}(b)$ *sa taille et* $\mathcal{B}(b)$ *sa base. L'intervalle correspondant dans la mémoire est :*

$$[\mathcal{B}(b), \mathcal{B}(b) + \mathcal{T}(b)[= [\mathcal{B}(b), \mathcal{B}(b) + \mathcal{T}(b) - 1]$$

Déf. II.3 (Bloc libre indivisible) *Un bloc libre est indivisible s'il ne peut pas être décomposé en blocs plus petits lors d'une allocation.*

Déf. II.4 (Blocs adjacents) *Deux blocs sont adjacents quand la fin d'un bloc se situe exactement avant le début de l'autre bloc, c'est-à-dire quand la base du second bloc est égale à la base du premier augmentée de sa taille.*

Déf. II.5 (Fusion de blocs) *La fusion de deux blocs adjacents a comme base la plus petite des bases des deux blocs, et comme taille la somme des tailles des deux blocs. Cette opération correspond à l'union des intervalles correspondant à chaque bloc.*

Déf. II.6 (Blocs disjoints) *Deux blocs sont disjoints si l'intersection des intervalles correspondants est vide.*

1.2 Représentation en PASCAL

Nous ferons l'hypothèse que les bases, respectivement les tailles, manipulées sont positives, respectivement strictement positives.

Un bloc est représenté par le type de base `BLOC`.

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 B_Creer(b,t :INTEGER) :BLOC ;` renvoie un bloc dont la base est `b` et la taille est `t`,
- `FUNCTION B_Base(b :BLOC) :INTEGER ;` renvoie la base du bloc `b`,
- `FUNCTION B_Taille(b :BLOC) :INTEGER ;` renvoie la taille du bloc `b`.

Nous allons maintenant définir plusieurs opérations sur les blocs.

1.3 Opérations sur la structure de bloc

La première opération est le test si deux blocs sont adjacents.

Question II.1 *Écrire en PASCAL une fonction* `adjacent(b1 :BLOC;b2 :BLOC) :BOOLEAN ;` *telle que l'appel* `adjacent(b1,b2)` *renvoie la valeur* `TRUE` *si le bloc* `b1` *est adjacent au bloc* `b2`*, et la valeur* `FALSE` *sinon.*

La seconde opération est la fusion de deux blocs.

Question II.2 *Écrire en PASCAL une fonction* `fusion(b1 :BLOC;b2 :BLOC) :BLOC ;` *telle que l'appel* `fusion(b1,b2)` *renvoie un bloc correspondant à la fusion de* `b1` *et* `b2` *si* `b1` *est adjacent au bloc* `b2`*, et renvoie le bloc de base* `-1` *et de taille* `-1` *sinon.*

2 Préliminaires : Codage binaire des nombres entiers positifs

Le codage binaire de taille t d'un nombre entier positif strictement inférieur à 2^t est représenté par la séquence $\langle b_t, \ldots, b_1 \rangle$ de chiffres binaires, ou bits (0 ou 1).

Notons que le premier chiffre est le bit de poids le plus fort et le dernier chiffre le bit de poids le plus faible. Ceci est essentiel pour la question II.11.

2.1 Définitions

Déf. II.7 (Codage binaire d'un nombre entier positif) *Soit un nombre entier positif* $n \in \mathbb{N}$*, le codage binaire unique de taille* t *du nombre entier positif* n *(avec* $n < 2^t$*) est la séquence* $\langle b_t, \ldots, b_1 \rangle$ *telle que :*

$$\forall i, 1 \leq i \leq t \Rightarrow b_i \in \{0, 1\}$$
$$n = \sum_{i=1}^{t} b_i \times 2^{i-1}$$

Question II.3 *Calculer la valeur de l'indice du chiffre binaire de valeur* 1 *de poids le plus fort du codage binaire du nombre entier positif* n *en fonction de la valeur de* n*.*

Question II.4 *Montrer que tout nombre entier strictement positif s'écrit de manière unique sous la forme du produit d'un nombre entier positif impair et d'une puissance de* 2*.*

2.2 Représentation en PASCAL

Le code binaire d'un nombre entier positif est représenté par le type de base `BINAIRE` correspondant à une liste de `INTEGER`.

Son parcours sera donc effectué de la même manière que celui d'une liste.

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 :

- `NIL` représente la liste vide de bits ;
- `FUNCTION E_Insertion(v :INTEGER;l :BINAIRE) :BINAIRE ;` renvoie une liste de bits composée d'un premier bit `v` et du reste de la liste contenu dans `l` ;
- `FUNCTION E_Premier(l :BINAIRE) :INTEGER ;` renvoie le premier bit de la liste `l`. Cette liste ne doit pas être vide ;
- `FUNCTION E_Reste(l :BINAIRE) :BINAIRE ;` renvoie le reste de la liste `l` privée de son premier bit. Cette liste ne doit pas être vide.

2.3 Nombre de bits nécessaire pour un codage

La première opération réalisée est le calcul du nombre de bits nécessaire pour le codage d'un nombre entier positif au format binaire.

Question II.5 *Écrire en PASCAL une fonction* `nombre(e :INTEGER) :INTEGER ;` *telle que l'appel* `nombre(e)` *renvoie le nombre de chiffres binaires nécessaires pour coder le nombre entier positif* `e` *au format binaire. Cette fonction pourra être récursive ou faire appel à des fonctions auxiliaires récursives.*

2.4 Codage d'un nombre entier positif au format binaire

La seconde opération réalisée est le codage d'un nombre entier positif au format binaire.

Question II.6 *Écrire en PASCAL une fonction* `codage(n :INTEGER) :BINAIRE ;` *telle que l'appel* `codage(n)` *renvoie une liste d'entiers composée des coefficients* $b_t, \ldots, b_1$ *correspondant au codage binaire de* n*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

3 Réalisation à base d'arbres binaires

L'utilisation de la structure d'arbre binaire pour représenter les blocs de mémoire permet de réduire la complexité en temps de calcul pour les opérations d'allocation et de libération.

L'ensemble de la mémoire, blocs libres et blocs alloués, est représentée par un arbre binaire dont les feuilles sont les blocs et les nœuds correspondent à une décomposition en deux parties égales de la mémoire.

Les blocs libres indivisibles sont donc de taille impaire.

3.1 Arbre binaire étiqueté

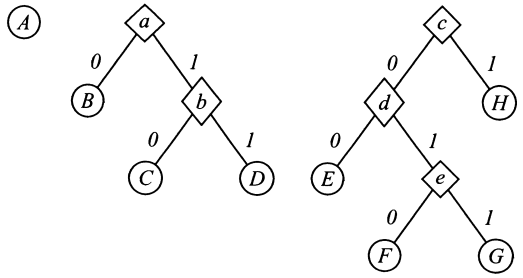
3.1.1 Définition

Déf. II.8 (Arbre binaire étiqueté) *Un arbre binaire étiqueté a est une structure qui peut :*
- *soit être une feuille qui peut contenir une étiquette,*
- *soit un nœud qui contient un sous-arbre gauche (noté $\mathcal{G}(a)$) et un sous-arbre droit (noté $\mathcal{D}(a)$) qui sont tous deux des arbres binaires étiquetés, et peut également contenir une étiquette.*

L'ensemble des feuilles, respectivement des nœuds, est noté $\mathcal{F}$, respectivement $\mathcal{N}$. L'ensemble des arbres est $\mathcal{A} = \mathcal{F} \cup \mathcal{N}$.

L'ensemble des feuilles, respectivement des nœuds, d'un arbre a est noté $\mathcal{F}(a)$, respectivement $\mathcal{N}(a)$.

Exemple II.1 (Arbres binaires étiquetés) *Voici trois exemples d'arbres binaires dont les nœuds, respectivement les feuilles, sont étiquetés par des minuscules, respectivement des majuscules, (les nœuds sont des losanges, les feuilles sont des cercles). Les arcs sont étiquetés par des chiffres binaires (0 pour le fils gauche et 1 pour le fils droit) qui seront utilisés pour représenter les chemins qui conduisent de la racine aux nœuds et aux feuilles dans l'arbre (voir la définition II.12) :*



3.1.2 Parcours gauche-droite d'un arbre binaire

Déf. II.9 (Parcours gauche-droite d'un arbre binaire) *Le parcours gauche-droite d'un arbre binaire a, noté $\mathcal{E}(a)$, construit la séquence des feuilles de l'arbre dans l'ordre de gauche à droite. Il est défini par les équations :*

$$\mathcal{E}(a) = \begin{cases} \langle a \rangle & \text{si } a \in \mathcal{F} \\ \langle f_1, \ldots, f_{m+n} \rangle & \text{si } a \in \mathcal{N} \\ & \text{et } \langle f_1, \ldots, f_m \rangle = \mathcal{E}(\mathcal{G}(a)) \\ & \text{et } \langle f_{m+1}, \ldots, f_{m+n} \rangle = \mathcal{E}(\mathcal{D}(a)) \end{cases}$$

Exemple II.2 (Parcours gauche-droite) *Les résultats du parcours gauche-droite des trois arbres de l'exemple II.1 sont respectivement $\langle A \rangle$, $\langle B, C, D \rangle$ et $\langle E, F, G, H \rangle$.*

3.1.3 Profondeur dans un arbre

Déf. II.10 (Profondeur dans un arbre) *La profondeur d'un élément e (nœud ou feuille) dans un arbre a est égale au nombre d'arcs entre la racine et cet élément dans l'arbre. Nous la noterons $\mathcal{P}(e, a)$.*

Exemple II.3 (Profondeur dans un arbre) *Les profondeurs des éléments A, b et e dans les trois arbres de l'exemple II.1 sont respectivement 0, 1 et 2.*

Déf. II.11 (Profondeur d'un arbre) *La profondeur d'un arbre a est égale au maximum de la profondeur de toutes ses feuilles. Nous la noterons $\mathcal{P}(a)$.*

Exemple II.4 (Profondeur d'un arbre) *Les profondeurs des arbres de l'exemple II.1 sont respectivement 0, 2 et 3.*

3.1.4 Chemin dans un arbre

Déf. II.12 (Chemin dans un arbre) *Le chemin d'un élément e (nœud ou feuille) dans un arbre a est la séquence des étiquettes des arcs entre la racine et cet élément dans l'arbre. Nous le noterons $\mathcal{C}(e, a)$.*

Notons que le chemin de la racine d'un arbre est vide.

Notons qu'un chemin correspond au codage binaire d'un nombre entier positif dont le bit de poids le plus fort est l'étiquette la plus proche de la racine.

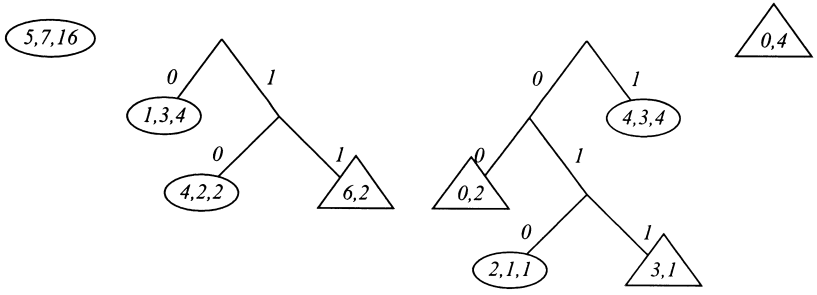
Exemple II.5 (Chemin d'un élément) *Les chemins des éléments A, C et G dans l'exemple II.1 sont respectivement $\langle\rangle$, $\langle 1, 0 \rangle$ et $\langle 0, 1, 1 \rangle$.*

3.2 Arbres binaires de blocs

Nous considérerons par la suite des arbres binaires dont les feuilles sont de deux types, celles qui représentent un bloc libre, et celles qui représentent un bloc alloué. Toutes les feuilles sont décorées par la base et la taille du bloc. Les feuilles allouées sont également décorées par la taille utilisée dans le bloc.

Déf. II.13 (Arbre binaire de blocs) *Un arbre binaire de blocs est un arbre binaire étiqueté dont :*
- *Chaque feuille f est :*
  - *soit une feuille libre qui contient deux étiquettes entières, sa base (notée $\mathcal{B}(f)$) et sa taille (notée $\mathcal{T}(f)$), l'ensemble des feuilles libres est noté $\mathcal{F}_L$,*
  - *soit une feuille occupée qui contient trois étiquettes entières, sa base (notée $\mathcal{B}(f)$), sa taille allouée (notée $\mathcal{O}(f)$) et sa taille totale (notée $\mathcal{T}(f)$), l'ensemble des feuilles occupées est noté $\mathcal{F}_O$ ($\mathcal{F} = \mathcal{F}_L \cup \mathcal{F}_O$).*
- *Les nœuds ne sont pas étiquetés.*
- *Les fils gauche et droit de chaque nœud sont des arbres binaires de blocs.*

Exemple II.6 (Arbres binaires de blocs) *Voici quatre exemples d'arbres binaires de blocs (les feuilles libres sont des triangles contenant la base puis la taille, les feuilles occupées sont des ovales contenant la base, puis la taille occupée puis la taille totale). Les arcs sont étiquetés par des chiffres binaires (0 pour le fils gauche et 1 pour le fils droit) qui seront utilisés pour représenter les chemins dans l'arbre (voir la définition II.12) :*



Déf. II.14 (Taille d'un arbre de blocs) *La taille totale, notée $\mathcal{T}(a)$, d'un arbre de blocs a est la somme de la taille de tous les blocs qui composent ses feuilles.*

Exemple II.7 (Taille d'un arbre de blocs) *Les tailles des arbres de l'exemple II.6 sont respectivement 16, 8, 8 et 4.*

Déf. II.15 (Taille maximum disponible) *La taille maximum disponible, notée $\mathcal{M}(a)$, d'un arbre de blocs a est la taille du plus grand des blocs libres qui composent ses feuilles.*

Exemple II.8 (Taille maximum disponible) *Les tailles maximum disponibles des arbres de l'exemple II.6 sont respectivement 0, 2, 2 et 4.*

Déf. II.16 (Base d'un arbre de blocs) *La base, notée $\mathcal{B}(a)$, d'un arbre de blocs a est la base du bloc contenu dans la feuille la plus à gauche de l'arbre.*

Exemple II.9 (Base d'un arbre de blocs) *Les bases des arbres de l'exemple II.6 sont respectivement 5, 1, 0 et 0.*

Déf. II.17 (Arbre d'allocation) *Un arbre d'allocation est un arbre de blocs qui vérifie les contraintes suivantes :*

C1 *Les fils gauche et droit d'un même nœud ne sont pas simultanément des feuilles libres.*

C2 *Les fils gauche et droit d'un même nœud sont de même taille.*

C3 *La base du fils droit d'un nœud est égale à la base de ce nœud augmentée de la taille du fils gauche.*

C4 *La taille allouée dans une feuille occupée est supérieure à la moitié de la taille totale de la feuille.*

Exemple II.10 (Arbres d'allocation) *Le premier et le deuxième arbres de l'exemple II.6 ne sont pas des arbres d'allocation car ils ne respectent pas les contraintes **C4** (le premier) et **C3** (le deuxième). Le troisième et le quatrième arbres de l'exemple II.6 sont des arbres d'allocation.*

Question II.7 *Exprimer la définition II.17 sous la forme d'une propriété $AA(e)$ qui indique que a est un arbre d'allocation. Vous pouvez utiliser pour cela les fonctions définies sur les arbres binaires de blocs.*

Déf. II.18 (Arbre de blocs dense) *Un arbre de blocs est dense si et seulement si chaque feuille dans un parcours gauche-droite est adjacente à sa voisine de droite.*

Question II.8 *Montrer qu'un arbre d'allocation est un arbre de blocs dense.*

Question II.9 *Soit a un arbre d'allocation, soit a', un sous-arbre de a, montrer que :* $\mathcal{T}(a) = \mathcal{T}(a') \times 2^{\mathcal{P}(a',a)}$.

Question II.10 *Montrer que les blocs libres indivisibles dans un arbre d'allocation doivent tous être de la même taille et de la même profondeur.*

Question II.11 *Montrer que la base d'un bloc b, distinct de a, dans un arbre d'allocation a est égale à la base de a augmentée du nombre entier positif dont le codage binaire est le chemin de b dans a que multiplie la taille de b.*

3.2.1 Représentation des arbres binaires de blocs en PASCAL

Pour simplifier le problème, nous ne considérerons que des mémoires dont la taille est une puissance de 2 (les blocs libres indivisibles seront donc de taille 1) et qui commencent à la base 0 donc des intervalles de la forme $[0, 2^n - 1]$ avec $n \in \mathbb{N}$.

Pour simplifier le problème, nous n'étiquetons pas les blocs avec leurs bases mais uniquement leurs tailles. Nous pouvons utiliser le chemin et la taille pour calculer la base d'un bloc en s'appuyant sur le résultat de la question II.11. Nous utiliserons donc par la suite le chemin au lieu de la base pour simplifier les fonctions.

Un arbre binaire de blocs est représenté par le type de base `ARBRE`.

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 Libre(t :INTEGER) :ARBRE ;` est une fonction qui renvoie une feuille libre dont la taille est `t`,
- `FUNCTION EstLibre(a :ARBRE) :BOOLEAN ;` est une fonction qui renvoie la valeur `TRUE` si l'arbre `a` est une feuille libre, et la valeur `FALSE` sinon ;
- `FUNCTION TailleLibre(a :ARBRE) :INTEGER ;` est une fonction qui renvoie la taille de la racine de l'arbre `a` si `a` est une feuille libre.
- `FUNCTION Occupe(to,tt :INTEGER) :ARBRE ;` est une fonction qui renvoie une feuille occupée dont la taille occupée est `to` et la taille totale est `tt`,
- `FUNCTION EstOccupe(a :ARBRE) :BOOLEAN ;` est une fonction qui renvoie la valeur `TRUE` si l'arbre `a` est une feuille occupée, et la valeur `FALSE` sinon ;
- `FUNCTION TailleOccupee(a :ARBRE) :INTEGER ;` est une fonction qui renvoie la taille occupée de la racine de l'arbre `a` si `a` est une feuille occupée.
- `FUNCTION TailleTotale(a :ARBRE) :INTEGER ;` est une fonction qui renvoie la taille totale de la racine de l'arbre `a` si `a` est une feuille occupée.
- `FUNCTION Noeud(fg :ARBRE ;fd :ARBRE) :ARBRE ;` est une fonction qui renvoie un nœud dont le fils gauche et le fils droit de la racine sont respectivement `fg` et `fd`,
- `FUNCTION EstNoeud(a :ARBRE) :BOOLEAN ;` est une fonction qui renvoie la valeur `TRUE` si l'arbre `a` est un nœud, et la valeur `FALSE` sinon ;
- `FUNCTION Gauche(a :ARBRE) :ARBRE ;` est une fonction qui renvoie le fils gauche de la racine de l'arbre `a` si `a` est un nœud ;
- `FUNCTION Droit(a :ARBRE) :ARBRE ;` est une fonction qui renvoie le fils droit de la racine de l'arbre `a` si `a` est un nœud.

Exemple II.11 *L'appel suivant*

```
Noeud(
   Noeud(
      Libre( 2 ),
      Noeud(
         Occupe( 1, 1),
         Libre( 1)
      )
   ),
   Occupe( 3, 4)
)
```

renvoie l'arbre binaire de blocs représenté graphiquement par le troisième arbre dans l'exemple II.6.

3.3 Opérations sur la structure d'arbre d'allocation

3.3.1 Espace libre maximum d'un arbre d'allocation

La première opération est le calcul de l'espace libre maximum disponible dans un arbre d'allocation.

Question II.12 *Écrire en PASCAL une fonction* `maximum(a :ARBRE) :INTEGER;` *telle que l'appel* `maximum(a)` *renvoie la taille du plus grand bloc libre de l'arbre d'allocation* `a`*, c'est-à-dire* $\mathcal{M}(a)$*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

3.3.2 Vérification de la structure d'ensemble d'allocation

La deuxième opération est la vérification qu'un arbre de blocs est un arbre d'allocation, c'est-à-dire respecte les contraintes de la définition II.17.

Question II.13 *Écrire en PASCAL une fonction* `verifier(a :ARBRE) :BOOLEAN;` *telle que l'appel* `verifier(a)` *sur un arbre d'allocation* `a` *renvoie la valeur* `TRUE` *si l'arbre* `a` *respecte les contraintes d'un arbre d'allocation, et la valeur* `FALSE` *sinon. L'algorithme utilisé ne devra parcourir qu'une seule fois l'arbre. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

3.3.3 Libération d'un bloc

La troisième opération est la libération d'un bloc dans un arbre d'allocation.

Question II.14 *Construire l'arbre d'allocation qui résulte de la libération du bloc dont la base est* 2 *dans le troisième arbre d'allocation de l'exemple II.6. Préciser les différentes étapes qui permettent de construire cet arbre.*

Question II.15 *Écrire en PASCAL une fonction* `liberer(c :BINAIRE;a :ARBRE) :ARBRE;` *telle que l'appel* `liberer(c,a)` *sur un arbre d'allocation* `a` *qui contient le bloc alloué de chemin* `c` *renvoie un arbre d'allocation contenant :*

- *les blocs alloués de* `a` *sauf le bloc de chemin* `c`*,*
- *le résultat de la fusion des blocs libres de* `a` *et du bloc de chemin* `c` *pour que le résultat respecte la structure d'arbre d'allocation.*
- *les blocs libres de* `a` *qui ne sont pas adjacents au bloc de chemin* `c`*,*
- *le bloc libéré de chemin* `c` *si aucun bloc libre de* `a` *ne lui est adjacent.*

Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.

Question II.16 *Donner des exemples de valeurs des paramètres* `c` *et* `a` *de la fonction* `liberer` *qui correspondent aux meilleur et pire cas en nombre d'appels récursifs effectués.*

Calculer une estimation de la complexité dans les meilleur et pire cas de la fonction `liberer` *en fonction de la profondeur de l'arbre d'allocation* `a`*. Cette estimation ne prendra en compte que le nombre d'appels récursifs effectués.*

3.3.4 Allocation d'un bloc

La quatrième opération est l'allocation d'un bloc dans un arbre d'allocation.

De nombreuses stratégies sont possibles pour choisir le bloc libre dans l'arbre qui sera utilisé parmi ceux dont la taille est supérieure ou égale à la taille demandée. Nous allons appliquer la stratégie consistant à choisir le bloc le plus petit parmi ceux-ci.

Lorsque le bloc choisi est de taille strictement supérieure à la taille demandée, ce bloc doit être décomposé en un sous-arbre qui contiendra le bloc alloué et un ou plusieurs blocs libres pour préserver la structure d'arbre d'allocation.

Question II.17 *Donner le chemin et construire l'arbre d'allocation qui résultent de l'allocation de la taille* 1 *dans le quatrième arbre d'allocation de l'exemple II.6. Donner les différentes étapes qui permettent de construire ce chemin et cet arbre.*

Le résultat d'une allocation est représenté par le type de base `RAA` qui contient un chemin et un arbre d'allocation.

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 RAA_Creer(c :BINAIRE;a :ARBRE) :RAA;` renvoie un résultat d'allocation dont le chemin est `c` et l'arbre d'allocation `a`,
- `FUNCTION RAA_Bloc(r :RAA) :BINAIRE;` renvoie le chemin contenu dans le résultat d'allocation `r`,
- `FUNCTION RAA_Arbre(r :RAA) :ARBRE;` renvoie l'arbre d'allocation contenu dans le résultat d'allocation `r`.

Question II.18 *Écrire en PASCAL une fonction* `allouer(t :INTEGER;a :ARBRE) :RAA;` *telle que l'appel* `allouer(t,a)` *sur un arbre d'allocation* `a` *renvoie un résultat d'allocation composé d'un chemin vers un bloc et d'un arbre d'allocation.*

- *Si* `a` *ne contient pas de bloc libre de taille supérieure ou égale à* `t`*, alors le chemin renvoyé contient uniquement le nombre entier* `-1`*, et l'arbre d'allocation renvoyé contient exactement les mêmes blocs que* `a`*.*
- *Si* `a` *contient au moins un bloc de taille supérieure ou égale à* `t` *alors, notons* `b` *un bloc parmi les plus petits blocs de* `a` *de taille supérieure ou égale à* `t`*,*
  - *le chemin renvoyé est celui d'un bloc alloué qui aura la même base que* `b` *et la taille* `t`*,*
  - *l'arbre d'allocation renvoyé contient :*
    - *les mêmes blocs que* `a` *sauf* `b`*,*
    - *les blocs qui résultent de la décomposition de* `b` *pour réaliser l'allocation en respectant les contraintes d'un arbre d'allocation, dont le bloc alloué.*

Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.

Question II.19 *Donner des exemples de valeurs des paramètres* `t` *et* `a` *de la fonction* `allouer` *qui correspondent aux meilleur et pire cas en nombre d'appels récursifs effectués.*

Calculer une estimation de la complexité dans les meilleur et pire cas de la fonction `allouer` *en fonction de la taille de l'arbre d'allocation* `a`*. Cette estimation ne prendra en compte que le nombre d'appels récursifs effectués.*

Question II.20 *Comparer les estimations des complexités en nombre d'appels récursifs dans les meilleur et pire cas pour la fonction* `allouer` *pour la structure d'ensemble d'allocation et la structure d'arbre d'allocation en fonction de la taille de la mémoire.*

Fin de l'énoncé