# Devoir surveillé n° 4 — Concours blanc — 3h

L'usage de la calculatrice est interdit. Les parties I et II sont indépendantes et sont **à rédiger sur des copies séparées**.

## I Le problème du sac à dos

Un alpiniste se prépare à un long périple dans l'Himalaya et se retrouve confronté à un problème crucial. En effet, il ne pourra porter au maximum dans son sac à dos qu'une quantité limitée de poids de corde. Malgré cette contrainte, il aimerait évidemment emporter avec lui le maximum de longueur possible. Il pourrait par exemple pouvoir porter au plus 10kg et disposer des cordes suivantes :

| type | poids | longueur |
|---|---|---|
| A | 6kg | 30m |
| B | 3kg | 14m |
| C | 4kg | 16m |
| D | 2kg | 9m |

I.1) On suppose qu'il y a dans le camp de base autant de cordes de chaque type que l'on veut. Quel choix de cordes maximise la longueur totale, tout en respectant la contrainte de poids maximum ?

I.2) On suppose maintenant qu'un seul exemplaire de chaque corde est disponible. Quel est alors le choix optimal dans ce cas ?

Formalisons un peu. On se donne un entier $W \geq 0$ représentant le poids maximal que l'on peut mettre dans le sac à dos, un entier $n \in \mathbb{N}$ représentant des objets numérotés de 1 à $n$ et deux vecteurs $(w_i)_{1\leq i\leq n} \in (\mathbb{N}^*)^n$ et $(v_i)_{1\leq i\leq n} \in (\mathbb{N}^*)^n$ représentant respectivement les poids et les valeurs des objets. Une façon de remplir le sac, appelée *choix*, est un vecteur $X = (x_i)_{1\leq i\leq n}$, où $x_i$ représente le nombre d'objets $i$ choisis.

La valeur totale d'un certain choix $X$ est alors

$$v(X) = \sum_{i=1}^{n} x_i v_i$$

et le poids associé à ce choix est

$$w(X) = \sum_{i=1}^{n} x_i w_i$$

Le problème est alors de trouver un choix qui maximise la valeur totale des objets mais dont la somme des poids des objets choisis ne dépasse pas la capacité du sac à dos :

$$v^* = \max_{\{X \,|\, w(X)\leq W\}} v(X)$$

On considère deux variantes possibles du problème :

- **Sans répétitions** : on ne dispose que d'un seul exemplaire de chaque objet. On modélise alors un choix comme un vecteur $X \in \{0,1\}^n$.
- **Avec répétitions** : les objets sont en quantités illimités et on peut prendre chaque objet autant de fois qu'on le souhaite. On modélise alors un choix comme un vecteur $X \in \mathbb{N}^n$.

On se propose d'utiliser la programmation dynamique pour résoudre les deux variantes de ce problème.

### I.1 Sac à dos sans répétitions

Pour $0 \leq w \leq W$ et $0 \leq j \leq n$ on définit $s(w,j)$ comme étant la valeur optimale du sous-problème de sac à dos sans répétitions pour un sac de capacité maximale $w$ en utilisant uniquement les objets compris entre 1 et $j$ inclus. On cherche donc $v^* = s(W,n)$.

I.3) Que vaut $s(w,j)$ si l'un des deux arguments est nul ?

I.4) Pour $j \geq 1$, en distinguant suivant s'il est mieux/possible de prendre l'objet $j$ ou non, exprimer la solution optimale $s(w,j)$ du sous-problème en fonction de solutions optimales de sous-sous-problèmes $s(w',j')$ avec $w' \leq w$ et/ou $j' \leq j$.

I.5) En utilisant une méthode tabulaire (*bottom-up*), écrire en CAML une fonction

```
sac_a_dos : int -> int vect -> int vect -> int
```

telle que `sac_a_dos capacite poids valeurs` renvoie la valeur optimale du problème du sac à dos sans répétitions pour un sac de capacité maximale `capacite`, un vecteur de poids `poids` de taille $n$ et un vecteur de valeurs `valeurs` de taille $n$ également. On supposera que les arguments vérifient les hypothèses du problème sans avoir à le vérifier. Attention aux indices.

I.6) On suppose que l'on a modifié la fonction précédente pour obtenir également en sortie une matrice de booléens `utilise` de taille $(W+1, n+1)$ telle que pour $0 \leq w \leq W$ et $0 \leq j \leq n$, `utilise.(w).(j)` vaut `true` si l'objet $j$ est utilisé pour atteindre $s(w, j)$ et `false` sinon [1]. Expliquer, sans l'implémenter, comment on pourrait obtenir cette matrice.

I.7) Écrire une fonction récursive

```
retrouve_solution :
  int -> int -> int vect -> bool vect vect -> int liste
```

telle que `retrouve_solution w j poids utilise` renvoie une liste constitué des (indices des) objets choisis pour obtenir $s(w, j)$. L'ordre des éléments dans la liste n'a pas d'importance.

I.8) Quelle est la complexité temporelle de la fonction `sac_a_dos` ? Quelle est sa complexité spatiale ? Expliquer, sans implémenter, comment on pourrait réduire sa complexité spatiale. Dans ce cas, pourra-t-on encore reconstruire la solution optimale ?

I.9) Quel pourrait-être l'avantage d'une implémentation récursive avec mémorisation (*top-down*) si par exemple la capacité ainsi que tous les poids des objets sont des multiples de 100 ?

## I.2 Sac à dos avec répétitions

On s'intéresse maintenant au problème du sac à dos avec répétitions, c'est-à-dire que l'on peut choisir un même objet autant de fois que nécessaire. On note alors, pour $0 \leq w \leq W$, $k(w)$ la valeur optimale du sous-problème de sac à dos avec répétitions pour un sac de capacité maximale $w$.

I.10) Exprimer $k(w)$ en fonction des solutions optimales de sous-problèmes.

I.11) Décrire un algorithme, sous forme de pseudo-code, permettant de résoudre ce problème.

I.12) Quelles sont les complexités spatiales et temporelles de cette approche ?

## I.3 Remarques culturelles

*Ce problème fait partie des 21 problèmes NP-complets identifiés par Richard Karp en 1972 comme étant parmi les plus difficiles en optimisation combinatoire. Un grand nombre d'autres problèmes NP-complets peuvent se ramener à ces 21 problèmes de base.*

I.13) Nous avons vu que les deux versions de ce problème peuvent être résolues par un algorithme de complexité $O(nW)$, ce semble bien polynomial et donc pas particulièrement difficile. En fait, on ne connaît aucun algorithme [2] dont la complexité soit polynomiale en la *taille de l'entrée*. Pouvez-vous expliquer pourquoi ceci n'est pas contradictoire avec la complexité que nous avons proposée ?

*Ce problème est à la base du premier algorithme de chiffrement asymétrique (dit aussi à « clé publique ») présenté par Martin Hellman, Ralph Merkle et Whitfield Diffie à l'université Stanford en 1976. On le retrouve également dans de nombreux domaines : dans les systèmes financiers, où étant donné un certain montant d'investissement il faut choisir les projets rapportant le plus d'argent possible ; pour la découpe de matériaux, afin de minimiser les pertes dues aux chutes ; dans le chargement de cargaisons (avions, camions, bateaux) ; et bien d'autres.*

## I.4 Autres variantes

I.14) On s'intéresse à une nouvelle variante ou les objets sont toujours en quantités limitées (variante sans répétitions) mais dans laquelle on peut éventuellement disposer de plusieurs exemplaires de chaque objet. Expliquer très succinctement comment on peut modifier le problème pour modéliser et résoudre cette nouvelle variante du problème de sac à dos.

I.15) (*Bonus ; pour les plus rapides uniquement : à traiter après la partie II* ) On suppose maintenant que l'on dispose de certains objets en quantités limitées alors que d'autres sont en quantités illimitées. Proposer une résolution de cette variante et donner la complexité de votre approche. On se contentera de donner les idées générales sans chercher à tout prix à justifier précisément. On cherchera, si possible, une complexité en $O(nW)$ également.

1. Lorsque les deux sont possibles on suppose avoir choisi arbitrairement.

2. Et on a de bonnes raisons de penser qu'il n'en existe pas.

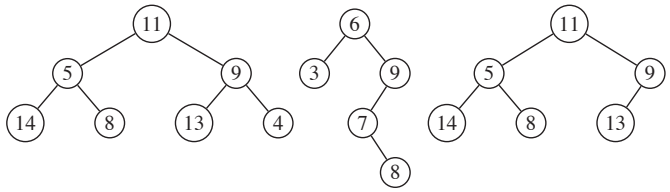
# II Arbres

## 1. Préliminaire : Arbre binaire d'entiers

### 1.1 Définition

**Déf. II.1 (Arbre binaire d'entiers)** Un arbre binaire d'entiers $a$ est une structure qui peut, soit être vide (notée $\emptyset$), soit être un nœud qui contient une étiquette entière (notée $\mathcal{E}(a)$), un sous-arbre gauche (noté $\mathcal{G}(a)$) et un sous-arbre droit (noté $\mathcal{D}(a)$) qui sont tous deux des arbres binaires d'entiers. L'ensemble des étiquettes de tous les nœuds de l'arbre $a$ est noté $\mathcal{C}(a)$.

**Exemple II.1 (Arbre binaire d'entiers)** Voici trois exemples d'arbres binaires étiquetés par des entiers (les sous-arbres vides qui sont fils gauche ou droit des nœuds ne sont pas représentés) :




### 1.2 Profondeur d'un arbre

**Déf. II.2 (Profondeur d'un arbre)** Les branches d'un arbre relient la racine aux sous-arbres vides. La profondeur d'un arbre $A$ est égale au nombre de liaisons entre nœuds de la branche la plus longue. Nous la noterons $\mid A \mid$. Nous associerons la profondeur $-1$ à l'arbre vide $\emptyset$.

**Exemple II.2 (Profondeurs)** Les profondeurs des trois arbres binaires de l'exemple II.1 sont respectivement 2, 3 et 2.

**Question II.1** *Donner une définition de la profondeur d'un arbre $a$ en fonction de $\emptyset$, $\mathcal{G}(a)$ et $\mathcal{D}(a)$.*

**Question II.2** *Soit un ensemble non vide d'étiquettes donné, quelle est la forme de l'arbre contenant ces étiquettes dont la profondeur est maximale ? Quelle est la forme de l'arbre contenant ces étiquettes dont la profondeur est minimale ?*

### 1.3 Arbres binaires complets

* Le niveau de profondeur p est l'ensemble des nœuds de profondeur p. La profondeur d'un nœud est le nombre de liaisons entre ce nœud et la racine (0 pour la racine donc).

**Déf. II.3 (Arbre binaire complet)** Un arbre binaire complet est un arbre binaire dont tous les niveaux* sont complets, c'est-à-dire que tous les nœuds d'un même niveau ont deux fils non vides sauf les nœuds du niveau le plus profond qui n'ont aucun fils (c'est-à-dire deux fils vides).

**Exemple II.3 (Arbre binaire complet)** Le premier arbre binaire de l'exemple II.1 est complet.

**Question II.3** *Montrer que, dans un arbre binaire complet non vide, le niveau de profondeur $p$ contient $2^p$ nœuds.*

**Question II.4** *Calculer le nombre $n$ de nœuds d'un arbre binaire complet non vide de profondeur $p$.*

**Question II.5** *En déduire la profondeur $p$ d'un arbre binaire complet non vide contenant $n$ éléments.*

## 2. Problème : Représentation de systèmes creux

La résolution numérique de problèmes en physique repose souvent sur la recherche de solutions pour un système linéaire qui résulte de la discrétisation du modèle mathématique continu utilisé pour représenter le problème. Lorsque les différentes parties du problème sont faiblement couplées, le système linéaire contient un grand nombre de valeurs nulles. Il est alors qualifié de système creux. Cette notion peut être généralisée à un système contenant un grand nombre de valeurs identiques. Nous appelons cette valeur identique, valeur par défaut des éléments. Pour les systèmes faiblement couplés évoqués précédemment, cette valeur par défaut est $0.0$.
L'objectif de ce problème est l'étude d'une représentation d'un vecteur creux par un arbre binaire en numérotant les nœuds par les indices des éléments du vecteur et en étiquetant les nœuds par les valeurs des éléments du vecteur différentes de la valeur par défaut. L'utilisation de cette structure permet de réduire le temps d'accès aux éléments du vecteur creux par rapport à l'utilisation naturelle d'une structure de liste. Par contre, cette structure utilise plus de mémoire que la liste mais moins que la représentation habituelle des vecteurs.
Un vecteur creux est représenté par un arbre binaire dont les noeuds sont soit vides, soit étiquetés par les valeurs des éléments contenus dans le vecteur creux qui sont différentes de la valeur par défaut, et les chemins de la racine de l'arbre aux nœuds sont extraits du codage binaire de l'indice de l'élément dont la valeur est contenue dans le nœud.

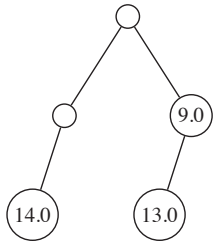
### 2.1 Arbre binaire partiel de réels

#### 2.1.1 Définition

Un arbre binaire partiel de réels est un arbre binaire de réels dont certains nœuds ne contiennent pas d'étiquettes.

**Déf. II.6 (Arbre binaire partiel de réels)** Un arbre binaire partiel de réels $a$ est une structure qui peut soit être vide (notée $\emptyset$), soit être un nœud qui contient une étiquette réelle (notée $\mathcal{E}(a)$), un sous-arbre gauche (noté $\mathcal{G}(a)$) et un sous-arbre droit (noté $\mathcal{D}(a)$) qui sont tous deux des arbres binaires partiels de réels, soit être une fourche qui est un nœud qui ne contient pas d'étiquette. L'ensemble des fourches de l'arbre $a$ est noté $\mathcal{F}(a)$. L'ensemble des nœuds de l'arbre $a$ qui ne sont pas des fourches est noté $\mathcal{N}(a)$.

**Exemple II.6 (Arbre binaire partiel de réels)** Voici un exemple d'arbre binaire partiel étiqueté par des réels (les sous-arbres vides des nœuds ne sont pas représentés) :



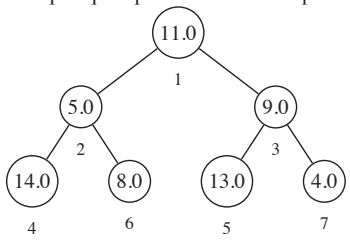
### 2.1.2 Numérotation hiérarchique des nœuds

Pour exploiter un arbre binaire partiel pour représenter un vecteur creux, il faut associer un indice à chaque nœud de l'arbre contenant une valeur.

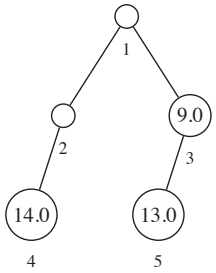
**Déf. II.7 (Numérotation hiérarchique des nœuds)** La numérotation hiérarchique des nœuds d'un arbre binaire consiste à associer le numéro $1$ à la racine de l'arbre puis à calculer les numéros des fils dans un nœud à partir du numéro de leur père. Soit un nœud de numéro $n$ à la profondeur $p$, le numéro de son fils gauche est $n + 2^p$, le numéro de son fils droit est $n + 2^{p+1}$.

Dans l'exemple suivant, le numéro de chaque nœud sera noté en-dessous de son étiquette.

**Exemple II.7** Voici un arbre complet qui représente un vecteur plein de 7 éléments.




Le vecteur plein correspondant est :
$\{1 \mapsto 11.0, 2 \mapsto 5.0, 3 \mapsto 9.0, 4 \mapsto 14.0, 5 \mapsto 13.0, 6 \mapsto 8.0, 7 \mapsto 4.0\}$.
Voici un arbre partiel qui représente un vecteur creux de 3 éléments avec une valeur par défaut $0.0$.




Le vecteur creux correspondant dont la valeur par défaut $0.0$ des éléments est explicitée :
$\{1 \mapsto 0.0, 2 \mapsto 0.0, 3 \mapsto 9.0, 4 \mapsto 14.0, 5 \mapsto 13.0, 6 \mapsto 0.0, 7 \mapsto 0.0\}$.

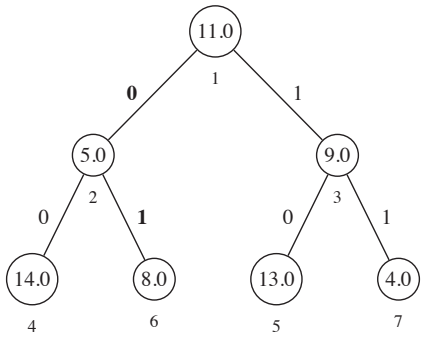
**Question II.11** *Montrer que l'ensemble des nœuds de profondeur $p$ est égal à l'intervalle $[2^p, 2^{p+1} - 1]$.*

**Question II.12** *Montrer que la numérotation de chaque nœud est unique.*

### 2.1.3 Occurrence d'un nœud dans un arbre

Pour accéder à un nœud, nous devons représenter le chemin dans l'arbre allant de la racine au nœud. Pour cela, nous étiquetons la liaison entre un nœud et son fils gauche par l'entier $0$ et la liaison entre un nœud et son fils droit par l'entier $1$. Nous appelons alors occurrence, ou chemin, du nœud de numéro $n$ la liste des étiquettes suivies pour aller de la racine à ce nœud. Dans l'exemple suivant, l'occurrence du nœud de numéro $6$ étiqueté par la valeur $8.0$ est $\langle 0, 1\rangle$.

**Exemple II.8**




Si l'occurrence du nœud de numéro $n$ situé à la profondeur $p$ est notée $\langle c_1, \ldots, c_p\rangle$ ($c_1$ étant le lien avec la racine), on remarque que l'occurrence du père de $n$ est $\langle c_1, \ldots, c_{p-1}\rangle$.

**Question II.13** *Soit un nœud de numéro $n$ situé à la profondeur $p$ et d'occurence $\langle c_1, \ldots, c_p\rangle$, montrer que :*

$$n = 2^p + \sum_{i=0}^{p-1} c_{i+1} \times 2^i$$

**Question II.14** *Exprimer la valeur du coefficient $c_i$ en fonction de $n$ et $i$.*

### 2.1.4 Représentation des arbres binaires partiels en CaML

Un arbre binaire partiel d'entiers est représenté par le type `arbre` dont la définition est :

```
type arbre =
| Vide
| Fourche of arbre * arbre
| Noeud of arbre * float * arbre;;
```

Dans l'appel `Noeud( fg, v, fd)`, les paramètres `fg`, `v` et `fd` sont respectivement le fils gauche, l'étiquette et le fils droit de la racine de l'arbre créé. Dans l'appel `Fourche( 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.9** L'expression suivante :

```
Fourche(
   Fourche( Noeud( Vide, 14.0, Vide), Vide),
   Noeud(
```

```
        Noeud( Vide, 13.0, Vide),
        9.0,
        Vide))
```

est alors associée à l'arbre binaire représenté graphiquement dans l'exemple II.6.

#### 2.1.5 Calcul du nombre de valeurs dans un arbre partiel

**Question II.15** *Écrire en CaML une fonction* `taille` *de type* `arbre -> int` *telle que l'appel* `(taille a)` *renvoie le nombre de nœuds étiquetés contenus dans l'arbre binaire partiel* `a`*, c'est-à-dire le cardinal de* $\mathcal{N}(a)$*. 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.*

#### 2.1.6 Calcul des bornes d'un arbre partiel

Une paire d'entiers est représentée par le type `paire` défini par :

```
type paire = int*int;;
```

Ceci est juste un nouveau nom pour le type int * int. La fonction bornes doit donc être de type arbre -> int * int.

**Question II.16** *Écrire en CaML une fonction* `bornes` *de type* `arbre -> paire` *telle que l'appel* `(bornes a)` *sur un arbre binaire partiel* `a` *non vide renvoie une paire d'entiers qui correspondent au plus petit, respectivement au plus grand, indice des valeurs contenues dans 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.*

#### 2.1.7 Remplacement d'une valeur dans un arbre partiel

**Question II.17** *Écrire en CaML une fonction* `remplacer` *de type* `int -> float -> arbre -> arbre` *telle que l'appel* `(remplacer i e a)` *sur un arbre binaire partiel* `a` *non vide contenant une valeur pour l'indice* `i` *renvoie un arbre binaire partiel contenant la valeur* `e` *à l'indice* `i` *et les mêmes valeurs que l'arbre* `a` *pour les autres indices que* `i`*. 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 Codage d'un vecteur creux par un arbre partiel

L'arbre partiel utilisé dans les questions précédentes correspond à des vecteurs creux dont les valeurs par défaut sont $0.0$. Ils ne contiennent donc que les valeurs différentes de $0.0$. Cette représentation peut être étendue pour représenter des vecteurs creux avec d'autres valeurs par défaut. Les étiquettes contenues dans l'arbre partiel correspondent alors aux indices et valeurs des éléments du vecteur creux qui sont différentes de la valeur par défaut du vecteur. La valeur par défaut doit faire partie explicitement de la structure du vecteur creux.
Pour améliorer les performances d'accès, la structure de vecteur creux contient en plus de l'arbre partiel et de la valeur par défaut, les minimum et maximum des indices des valeurs contenues dans le vecteur qui sont différentes de la valeur par défaut ainsi que le nombre de valeurs différentes de la valeur par défaut.
Soit $v$ un vecteur creux, nous noterons respectivement $\mathcal{V}_d(v)$, $\mathcal{V}_{min}(v)$, $\mathcal{V}_{max}(v)$, $\mathcal{V}_t(v)$ et $\mathcal{V}_a(v)$, la valeur par défaut, l'indice minimum, l'indice maximum, le nombre de valeurs différentes de la valeur par défaut et l'arbre partiel contenant les valeurs de $v$.

**Déf. II.8 (Vecteur creux bien formé)** Un vecteur creux implanté avec un arbre binaire partiel est bien formé si et seulement si :
– les indices minimum et maximum du vecteur creux correspondent aux bornes de l'arbre binaire partiel ;
– l'arbre binaire partiel ne contient pas la valeur par défaut du vecteur creux ;
– le nombre de valeurs contenues dans l'arbre binaire partiel est égal au nombre d'éléments différents de la valeur par défaut du vecteur creux ;
– l'arbre binaire partiel qui contient les valeurs ne doit pas contenir de fourches dont les sous-arbres gauche et droit sont tous deux vides.

**Question II.18** *Exprimer la définition II.8 sous la forme d'une propriété* $VBF(v)$ *qui indique que* $v$ *est un vecteur creux bien formé.*

Un vecteur creux est représenté par le type `vecteur` dont la définition est :

```
type vecteur = int * int * int * float * arbre;;
```

Encore une fois, juste un raccourci de notation de type.

Dans l'expression `( imin, imax, t, v, a)`, les paramètres `imin`, `imax`, `t`, `v` et `a` sont respectivement l'indice minimum, l'indice maximum et le nombre des éléments dont la valeur est différente de la valeur par défaut, ainsi que la valeur par défaut et l'arbre binaire partiel contenant les valeurs du vecteur.

**Question II.19** *Écrire en CaML une fonction* `valider` *de type* `vecteur -> bool` *telle que l'appel* `(valider v)` *renvoie la valeur* `true` *si le vecteur creux* `v` *est bien formé (c'est-à-dire si* $VBF(v)$*) et la valeur* `false` *sinon. L'algorithme utilisé ne devra parcourir qu'une seule fois l'arbre associé à* `v`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

**Question II.20** *Écrire en CaML une fonction* `lire` *de type* `int -> vecteur -> float` *telle que l'appel* `(lire i v)` *renvoie la valeur qui se trouve à l'indice* `i` *dans le vecteur creux* `v`*. L'algorithme utilisé ne devra parcourir qu'une seule fois l'arbre associé à* `v`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

**Question II.21** *Écrire en CaML une fonction* `ecrire` *de type* `int -> float -> vecteur -> vecteur` *telle que l'appel* `(ecrire i e v)`*, avec la valeur de* `e` *différente de la valeur par défaut de* `v`*, renvoie un vecteur creux contenant la valeur* `e` *à l'indice* `i` *et les mêmes valeurs que le vecteur creux* `v` *pour les autres indices que* `i`*. L'algorithme utilisé ne devra parcourir qu'une seule fois l'arbre associé à* `v`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

**Question II.22** *Écrire en CaML une fonction* `somme` *de type* `vecteur -> vecteur -> vecteur` *telle que l'appel* `(somme v1 v2)` *renvoie un vecteur creux dont les éléments ont*