CCP 2015 (extrait)

6 novembre 2017

2 Problème : le tri par tas

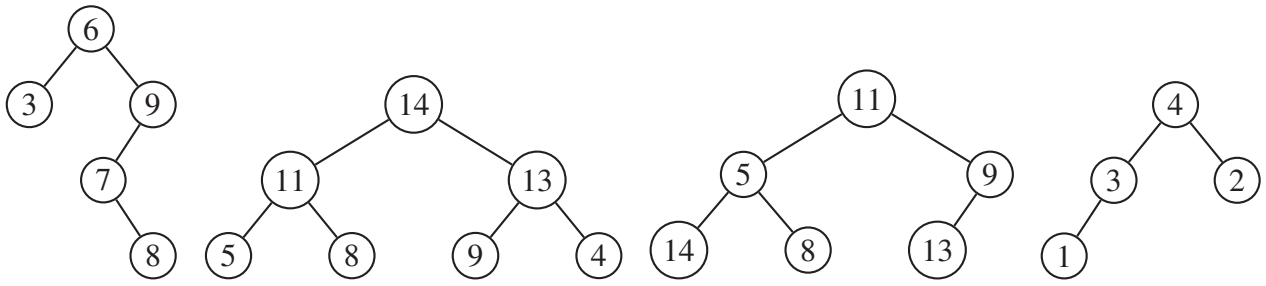
La complexité de l'algorithme de tri à bulles étudié dans l'exercice précédent ne permet pas de l'utiliser pour de gros volumes de données. Ce problème étudie un algorithme de tri plus performant basé sur la structure d'arbre en tas, c'est-à-dire d'arbre binaire parfait partiellement ordonné. Nous considérerons une implantation arborescente en langage CaML et une implantation sous la forme de tableau en langage Python.

2.1 Arbre binaire d'entiers

Déf. III.2 (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. La taille de l'arbre a, notée $\mid a \mid$ est le nombre de nœuds de l'arbre a.
Cette définition s'exprime sous la forme de la propriété $\mathcal{B}(a)$ qui caractérise les arbres binaires d'entiers a :

$$\mathcal{B}(a) \equiv (a = \emptyset) \vee \big(a \neq \emptyset \wedge \mathcal{B}(\mathcal{G}(a)) \wedge \mathcal{B}(\mathcal{D}(a)) \wedge \mathcal{E}(a) \in \mathbb{N}\big).$$

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



2.1.1 Implantation en langage CaML

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

```
type arbre =
 | Vide
 | Noeud of arbre * int * 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 binaire d'entiers créé.

Exemple III.2 L'expression suivante :

```
Noeud(
   Noeud( Vide, 3, Vide)
   6,
   Noeud(
      Noeud(
         Vide,
         7,
         Noeud( Vide, 8, Vide)),
      9,
      Vide))
```

est alors associée au premier arbre binaire représenté graphiquement dans **l'exemple III.1, page 8**.

Question III.7 *Donner l'expression en langage CaML qui correspond au quatrième arbre binaire de **l'exemple III.1, page 8**.*

2.1.2 Hauteur dans un arbre binaire

Déf. III.3 (Hauteur dans un arbre binaire) Soit a un arbre binaire et n un nœud de a. La hauteur de n dans a est égale au nombre de nœuds du chemin sans cycle le plus long reliant n à un sous-arbre vide. Nous la noterons $\eta(n)$. Si n est la racine de l'arbre ($n = a$), il s'agit alors de la hauteur de l'arbre. Nous associerons la hauteur 0 à l'arbre binaire vide $\emptyset$.

Exemple III.3 (Hauteurs) Les hauteurs des quatre arbres binaires de **l'exemple III.1, page 8** sont respectivement 4, 3, 3 et 3.

Question III.8 *Donner une définition mathématique de la hauteur $\eta(a)$ d'un arbre binaire a en fonction de $\emptyset$, $\mathcal{G}(a)$ et $\mathcal{D}(a)$.*

Question III.9 *Soit un ensemble non vide d'étiquettes donné, quelle est la structure de l'arbre binaire contenant ces étiquettes dont la hauteur est maximale ? Quelle est la structure de l'arbre binaire contenant ces étiquettes dont la hauteur est minimale ? Justifier vos réponses.*

2.1.3 Profondeur d'un nœud et niveau dans un arbre binaire

Déf. III.4 (Profondeur d'un nœud, niveau dans un arbre binaire) Soit un arbre binaire a contenant un nœud n, la profondeur du nœud n dans a notée $\pi(n)$ est définie par :

$$\pi(n) = \eta(a) - \eta(n).$$

Un niveau dans un arbre binaire a est la séquence de gauche à droite des nœuds de même profondeur dans a.

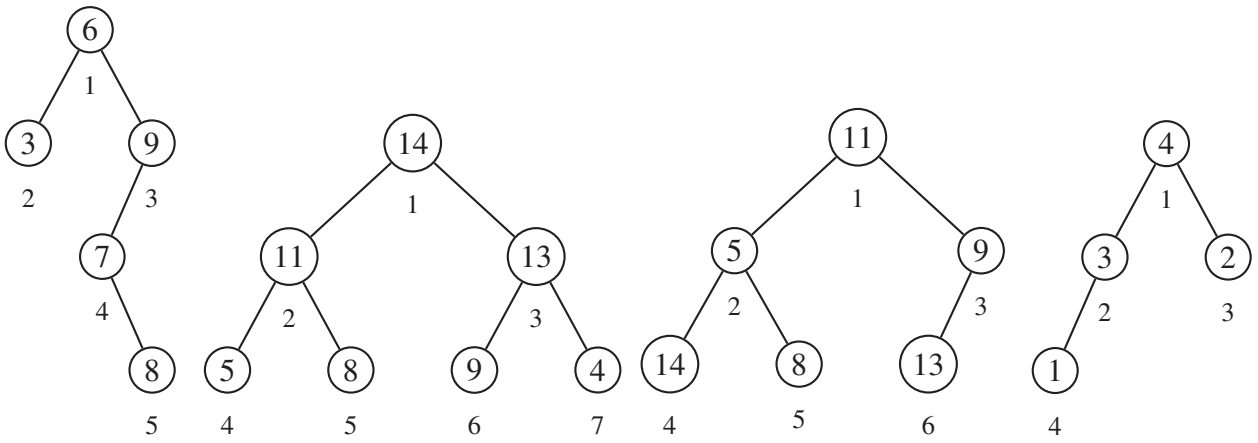
Exemple III.4 (Profondeurs) Les étiquettes des nœuds de même profondeur dans les quatre arbres de **l'exemple III.1, page 8** sont :

Profondeur	Arbre 1	Arbre 2	Arbre 3	Arbre 4
0	6	14	11	4
1	3, 9	11, 13	5, 9	3, 2
2	7	5, 8, 9, 4	14, 8, 13	1
3	8			

2.1.4 Numérotation hiérarchique des nœuds d'un arbre binaire

La numérotation hiérarchique des nœuds d'un arbre binaire a consiste à associer à chaque nœud un numéro compris entre 1 et $\mid a \mid$ par un parcours en largeur partant de la racine (numéro 1) et en parcourant chaque niveau de gauche à droite jusqu'au dernier nœud : le plus profond et le plus à droite (numéro $\mid a \mid$). Nous noterons $\mathcal{N}_i(a)$ le nœud de l'arbre binaire a de numéro i avec $i \in [1, \mid a \mid]$. Dans les exemples suivants, le numéro de chaque nœud sera noté en-dessous de son étiquette.

Exemple III.5 (Numérotation hiérarchique) La numérotation hiérarchique des nœuds des quatre arbres de **l'exemple III.1, page 8** produit les arbres numérotés suivants (les sous-arbres vides qui sont fils gauche ou droit des nœuds ne sont pas représentés) :



Question III.10 *Ecrire en CaML une fonction* `lire` *dont le type est* `int -> arbre -> int` *telle que l'appel* `(lire i a)` *sur l'arbre binaire d'entiers* `a` *avec* `i` $\in [1, \mid$ `a` $\mid]$ *doit renvoyer l'entier* $\mathcal{E}(\mathcal{N}_{\texttt{i}}(\texttt{a}))$. *Cette fonction devra au plus parcourir une seule fois chaque élément de l'arbre* `a`. *Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

2.2 Arbre binaire partiellement ordonné d'entiers

Déf. III.5 (Arbre binaire partiellement ordonné d'entiers) Un arbre binaire d'entiers a est partiellement ordonné si :

- les fils gauche et droit de a sont des arbres binaires partiellement ordonnés d'entiers ;
- les étiquettes de tous les nœuds composant les fils gauche et droit de a sont inférieures ou égales à l'étiquette de a.

Cette définition s'exprime sous la forme de la propriété $\mathcal{O}(a)$ qui caractérise les arbres binaires partiellement ordonnés d'entiers a :

$$\mathcal{O}(a) \equiv (a = \emptyset) \vee (a \neq \emptyset \wedge \mathcal{O}(\mathcal{G}(a)) \wedge \mathcal{O}(\mathcal{D}(a)) \wedge \mathcal{E}(\mathcal{G}(a)) \leq \mathcal{E}(a) \wedge \mathcal{E}(\mathcal{D}(a)) \leq \mathcal{E}(a)).$$

Exemple III.6 (Arbres binaires partiellement ordonnés d'entiers) Le deuxième et le quatrième arbres de **l'exemple III.1, page 8** sont des arbres binaires partiellement ordonnés d'entiers.

Question III.11 *Ecrire en CaML une fonction* `verifier` *dont le type est* `arbre -> bool` *telle que l'appel* `(verifier a)` *sur l'arbre binaire d'entiers* `a` *renvoie la valeur* `true` *si* $\mathcal{O}$(`a`) *et la valeur* `false` *sinon. Cette fonction devra au plus parcourir une seule fois chaque nœud de l'arbre* `a`. *Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

2.3 Arbres binaires complets

Déf. III.6 (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).
Cette définition s'exprime sous la forme de la propriété $\mathcal{C}_n(a)$ qui caractérise les arbres binaires complets a de hauteur n ($\eta(a) = n$) :

$$\mathcal{C}_n(a) \equiv (a = \emptyset \wedge n = 0) \vee (a \neq \emptyset \wedge n \neq 0 \wedge \mathcal{C}_{n-1}(\mathcal{G}(a)) \wedge \mathcal{C}_{n-1}(\mathcal{D}(a))).$$

Exemple III.7 (Arbre binaire complet) Le deuxième arbre de **l'exemple III.1, page 8** est complet.

Question III.12 *Montrer que, dans un arbre binaire complet non vide a, le niveau de profondeur p contient 2^p nœuds.*

Question III.13 *Calculer le nombre n de nœuds d'un arbre binaire complet non vide a de hauteur $p = \eta(a)$.*

Question III.14 *En déduire la hauteur $p = \eta(a)$ d'un arbre binaire complet non vide a contenant n éléments ($n = \mid a \mid$).*

2.4 Arbres binaires parfaits

Déf. III.7 (Arbre binaire parfait) Un arbre binaire parfait est un arbre binaire dont tous les niveaux sont complets sauf le niveau le plus profond qui peut être incomplet auquel cas ses nœuds sont alignés à gauche de l'arbre.
Cette définition s'exprime sous la forme de la propriété $\mathcal{P}_n(a)$ qui caractérise l'arbre binaire parfait a de hauteur n :

$$\mathcal{P}_n(a) \equiv \mathcal{C}_n(a) \vee a \neq \emptyset \wedge n \neq 0 \wedge \left\{ \begin{array}{ll} & n \neq 1 \wedge \mathcal{P}_{n-1}(\mathcal{G}(a)) \wedge \mathcal{C}_{n-2}(\mathcal{D}(a)) \\ \vee & n \neq 1 \wedge \mathcal{C}_{n-1}(\mathcal{G}(a)) \wedge \mathcal{C}_{n-2}(\mathcal{D}(a)) \\ \vee & \mathcal{C}_{n-1}(\mathcal{G}(a)) \wedge \mathcal{P}_{n-1}(\mathcal{D}(a)) \end{array} \right.$$

Exemple III.8 (Arbres binaires parfaits) Le troisième et le quatrième arbres de **l'exemple III.1, page 8** sont parfaits. Le deuxième est également parfait car il est complet.

Soit le type énuméré `categorieArbre` en langage CaML distinguant les arbres binaires complets, parfaits et quelconques :

```
type categorieArbre = Complet| Parfait| Quelconque;;
```

Question III.15 *Ecrire en CaML une fonction* `analyser` *dont le type est* `arbre -> int * categorieArbre` *telle que l'appel* `(analyser a)` *sur l'arbre binaire* `a` *renvoie une paire contenant la hauteur $\eta($`a`$)$ de l'arbre* `a` *ainsi que la valeur* `Complet` *si $\mathcal{C}_{\eta(\mathtt{a})}(\mathtt{a})$, sinon la valeur* `Parfait` *si $\mathcal{P}_{\eta(\mathtt{a})}(\mathtt{a})$ sinon la valeur* `Quelconque`*. Cette fonction devra au plus parcourir une seule fois chaque nœud de l'arbre* `a`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

Question III.16 *Soit un nœud de numéro n dans le niveau de profondeur p d'un arbre binaire parfait, calculer le nombre de nœuds qui se trouvent à sa gauche dans le niveau de profondeur p.*

Question III.17 *Dans le niveau de profondeur $p+1$ d'un arbre binaire parfait, quel est le nombre de nœuds qui se trouvent à la gauche des fils du nœud de numéro n (n faisant partie du niveau de profondeur p).*

Question III.18 *Soit un nœud de numéro n d'un arbre binaire parfait, calculer les numéros de ses fils gauche et droit.*

Question III.19 *Déduire de la question précédente, le numéro du père du nœud de numéro n dans un arbre binaire parfait.*

Question III.20 *Ecrire en CaML une fonction* `lire` *dont le type est* `int -> arbre -> int` *telle que l'appel* `(lire i a)` *sur l'arbre binaire parfait* `a` *avec* `i` $\in [1, \mid$ `a` $\mid]$ *doit renvoyer l'entier* $\mathcal{E}(\mathcal{N}_{\texttt{i}}(\texttt{a}))$. *Cette fonction devra au plus parcourir une seule fois chaque élément de la branche qui conduit de la racine de* `a` *au nœud de numéro* `i`. *Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

Soit le programme en langage CaML :

```
let construire l =
  let rec aux1 l a =
    match l,a with
      | ([],_) -> (a,l)
      | (t::q,Vide) -> (Noeud(Vide,t,Vide),q)
      | (_,Noeud(g,v,d)) ->
        match (aux1 l g) with
          | (rga,[]) ->
            ((Noeud(rga,v,d)),[])
          | (rga,rgl) ->
            (match (aux1 rgl d) with
              | (rda,rdl) ->
                ((Noeud(rga,v,rda)),rdl))
  in
  let rec aux2 l a =
    match (aux1 l a) with
      | (ra,[]) -> ra
      | (ra,rl) -> (aux2 rl ra)
  in
  (aux2 l Vide);;

let exemple = [ 1; 2; 3; 4; 5; 6];;
```

Question III.21 *Détailler les étapes du calcul de* `(construire exemple)` *en précisant pour chaque appel aux fonctions* `construire`, `aux1` *et* `aux2`, *la valeur du paramètre et du résultat.*

Question III.22 *Soient les séquences d'entiers* `s` *et* `r` *avec* $dom(\texttt{s}) = [1, m]$ *et* $dom(\texttt{r}) = [1, n]$*, les arbres binaires d'entiers* `a` *et* `b` *tels que* $\mathcal{C}_p(\texttt{a})$ *et* `(b,r)` *est la paire renvoyée par* `(aux1 s a)`*, montrer que :*

(i) $\forall i, 1 \leq i < 2^p, \mathcal{E}(\mathcal{N}_i(\texttt{b})) = \mathcal{E}(\mathcal{N}_i(\texttt{a}))$ *;*

(ii) $m = 0 \implies (\mathcal{C}_p(b) \wedge n = 0)$ *;*

(iii) $1 \leq m < 2^p \implies (\mathcal{P}_{p+1}(b) \wedge n = 0 \wedge \forall i \in dom(s), \mathcal{E}(\mathcal{N}_{i+2^p-1}(\texttt{b})) = s_i)$ *;*

(iv) $m \geq 2^p \implies (\mathcal{C}_{p+1}(b) \wedge n = m - 2^p \wedge \forall i \in dom(r), r_i = s_{i+2^p-1})$.

Question III.23 *Soit la séquence* `s` *avec* $dom(\texttt{s}) = [1, m]$*, soit l'arbre binaire d'entiers* `a`*, tels que* `a = (construire s)`*, montrer que :*

(i) $\mathcal{P}_p(\texttt{a})$ *;*

(ii) $\mid a \mid = m$ *;*

(iii) $2^{p-1} \leq m < 2^p$ *;*

(iv) $\forall i \in dom(s), \mathcal{E}(\mathcal{N}_i(\texttt{a})) = s_i$.

Question III.24 *Montrer que le calcul des fonctions* `aux1`*,* `aux2` *et* `construire` *se termine quelles que soient les valeurs de leurs paramètres respectant le type des fonctions.*

Question III.25 *Donner des exemples de valeurs du paramètre* `s` *de la fonction* `construire` *qui correspondent au pire cas en nombre d'appels récursifs effectués.*
Montrer que la complexité de la fonction `construire` *en fonction de la taille* n *des séquences données en paramètre est de* $O(n^2)$*. Cette estimation ne prend en compte que le nombre d'appels récursifs effectués.*

2.5 Arbres en tas

Déf. III.8 (Arbre en tas) Un arbre en tas est un arbre binaire parfait partiellement ordonné.
Cette définition s'exprime sous la forme de la propriété $\mathcal{T}_n(a)$ qui caractérise les arbres en tas a de hauteur n :

$$\mathcal{T}_n(a) \equiv \mathcal{P}_n(a) \wedge \mathcal{O}(a).$$

Exemple III.9 (Arbres en tas) Le deuxième et le quatrième arbres binaires de **l'exemple III.1, page** sont en tas.

Lorsqu'un arbre binaire parfait n'est pas partiellement ordonné, les étiquettes des nœuds peuvent être permutées pour obtenir un arbre en tas sans changer la structure d'arbre binaire parfait.

2.5.1 Implantation en langage CaML

Question III.26 *Ecrire en CaML une fonction* `placer` *dont le type est* `arbre -> int -> arbre -> arbre` *telle que l'appel* `(placer g v d)` *sur l'entier* `v` *et les arbres en tas* `g` *et* `d` *tels que* $\eta(\texttt{g}) = \eta(\texttt{d})$ *ou* $\eta(\texttt{g}) = \eta(\texttt{d}) + 1$ *(*η *est* ***définie page 9****), renvoie un arbre en tas contenant les même étiquettes que* `g` *et* `d` *ainsi que l'étiquette* `v`*. Cette fonction devra au plus parcourir une branche de* `g` *ou de* `d`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

Question III.27 *Proposer la modification la plus simple possible de la fonction* `construire` *étudiée précédemment* ***page 12*** *pour que l'arbre résultat ait une structure d'arbre en tas.*

2.5.2 Implantation par un tableau en langage Python

La structure d'arbre parfait peut être implantée très efficacement par un tableau contenant les étiquettes de l'arbre selon une numérotation hiérarchique des nœuds de l'arbre. Les nœuds de l'arbre seront désignés par leur numéro. L'indice le plus petit de la séquence doit être au moins égal à 1. Vous noterez que, dans le cas où l'indice le plus petit de la séquence est strictement plus grand que 1, toutes les cases du tableau ne correspondent pas à des nœuds de l'arbre. Il est également possible de préciser l'indice de la dernière case utilisée pour représenter le tableau. Alors, les cases suivantes ne correspondent pas à des nœuds.

Exemple III.10 (Arbres binaires parfaits représentés par un tableau) Les deuxième, troisième et quatrième arbres de **l'exemple III.1, page 8** sont parfaits. Ils sont représentés par les tableaux suivants (les entiers des lignes supérieures sont les numéros des nœuds et les entiers des lignes inférieures correspondent aux valeurs de leurs étiquettes) :

1	2	3	4	5	6	7
14	11	13	5	8	9	4

1	2	3	4	5	6
11	5	9	14	8	13

1	2	3	4
4	3	2	1

Pour le premier tableau, si l'indice correspondant à la racine de l'arbre parfait est le 3, alors seules les cases 3, 6 et 7 correspondent à des nœuds de l'arbre (**exemple III.5, page 10**).

Question III.28 *Soient :*

- `a` *une séquence d'entiers avec* $dom(\texttt{a}) = [1, n]$ *;*
- `r` *et* `f` *des entiers avec* $1 \leq \texttt{r} \leq \texttt{f} \leq n$ *;*

tels que :

- `a` *représente un arbre binaire parfait avec* `f` *l'indice du dernier nœud ;*
- *si l'étiquette* $\mathcal{E}(\mathcal{N}_{\texttt{r}}(\texttt{a}))$ *est remplacée par le maximum de* $\mathcal{E}(\mathcal{G}(\mathcal{N}_{\texttt{r}}(\texttt{a})))$ *et* $\mathcal{E}(\mathcal{D}(\mathcal{N}_{\texttt{r}}(\texttt{a})))$ *alors* $\mathcal{N}_{\texttt{r}}(\texttt{a})$ *est un arbre en tas.*

Ecrire en Python une procédure `placer(r,a,f)` *qui permute le contenu de certaines cases du tableau* `a` *pour transformer l'arbre binaire parfait contenu dans* `a` *avant l'appel dont le dernier nœud est de numéro* `f`*, en un arbre binaire parfait contenu dans* `a` *après l'appel tel que* $\mathcal{N}_{\texttt{r}}(\texttt{a})$ *soit un arbre en tas dont le dernier nœud est de numéro* `f`*. Seules les cases de* $\mathcal{N}_{\texttt{r}}(\texttt{a})$ *peuvent être modifiées. Cette procédure est donc telle que :*

- a *est le contenu de* `a` *avant l'exécution de l'appel* `placer(r,a,f)` *;*
- a' *est le contenu de* `a` *après l'exécution de l'appel* `placer(r,a,f)` *;*
- $\mathcal{P}_m(a) \wedge \mathcal{P}_m(a') \wedge \mathcal{T}_p(\mathcal{N}_{\texttt{r}}(a'))$ *;*
- $2^{m-1} \leq n < 2^m \wedge p + \texttt{r} \in \{m-1, m\}$ *;*
- $\langle a_i \rangle_{i=1}^{\texttt{r}-1} = \langle a'_i \rangle_{i=1}^{\texttt{r}-1}$ *;*
- $\{a_i\}_{i=\texttt{r}}^{\texttt{f}} = \{a'_i\}_{i=\texttt{r}}^{\texttt{f}}$ *;*
- $\langle a_i \rangle_{i=\texttt{f}+1}^{n} = \langle a'_i \rangle_{i=\texttt{f}+1}^{n}$.

Cette fonction devra au plus parcourir une branche de l'arbre binaire parfait représenté par `a`*. Cette fonction ne devra pas être récursive ni faire appel à des fonctions auxiliaires récursives.*

Question III.29 *Ecrire en Python une procédure* `entasser(a)` *qui permute le contenu de certaines cases du tableau d'entiers* `a` *pour transformer l'arbre binaire parfait contenu dans* `a` *avant l'appel en un arbre en tas contenu dans* `a` *après l'appel. Cette procédure est donc telle que :*

- `a` *est une séquence d'entiers avec* $dom(\texttt{a}) = [1, n]$ *;*
- a *est le contenu de* `a` *avant l'exécution de l'appel* `entasser(a)` *;*
- a' *est le contenu de* `a` *après l'exécution de l'appel* `entasser(a)` *;*
- $\mathcal{P}_m(a)$ *;*
- $\mathcal{T}_m(a')$ *;*
- $2^{m-1} \leq n < 2^m$ *;*
- $codom(a) = codom(a')$.

Cette fonction ne devra pas être récursive ni faire appel à des fonctions auxiliaires récursives.

2.6 Tri par tas

La structure d'arbre en tas permet d'implanter un algorithme de tri efficace appelé tri par tas. La structure d'arbre binaire partiellement ordonné assure que l'étiquette de la racine de l'arbre en tas contient la plus grande étiquette du tas. L'algorithme répète l'étape suivante jusqu'à ce que le tas soit vide :

- l'étiquette de la racine est placée à la fin de la séquence triée ;
- le dernier nœud du tas selon la numérotation hiérarchique est enlevé du tas. Son étiquette remplace celle de la racine. L'arbre binaire ainsi obtenu est parfait. Les sous-arbres gauche et droit de sa racine sont des arbres en tas ;
- l'arbre binaire parfait ainsi obtenu est ensuite converti en arbre en tas en utilisant la fonction `entasser`.

Question III.30 *Détailler les étapes de l'algorithme du tri par tas pour le quatrième arbre de* ***l'exemple III.1, page 8****. Pour chaque étape, donner les représentations du tas sous les formes d'arbre et de tableau.*

2.6.1 Implantation en langage Python

L'utilisation d'un tableau pour représenter un tas permet de permuter l'étiquette de la racine et celle du dernier nœud en une seule étape et de construire le tableau trié à partir de la fin.

Question III.31 *Ecrire en Python une procédure* `trier(s)` *qui permute le contenu de certaines cases du tableau* `s` *pour trier ce tableau. Cette procédure est donc telle que :*

- `s` *est une séquence d'entiers avec* $dom(\texttt{s}) = [1, n]$ *;*
- s *est le contenu de* `s` *avant l'exécution de l'appel* `trier(s)` *;*
- s' *est le contenu de* `s` *après l'exécution de l'appel* `trier(s)` *;*
- $codom(s) = codom(s')$ *;*
- $\forall i \in [1, n[, s'_i \leq s'_{i+1}$.

Cette fonction ne devra pas être récursive ni faire appel à des fonctions auxiliaires récursives.

2.6.2 Implantation en langage CaML

Le langage CaML ne permet pas de permuter en une seule étape les étiquettes de la racine et du dernier nœud. Cette permutation sera donc combinée avec les opérations `placer` et `entasser` qui réorganisent l'arbre binaire pour préserver la structure de tas.

Question III.32 *Ecrire en CaML une fonction* `remonter` *dont le type est* `int -> arbre -> arbre` *telle que l'appel* `(remonter i a)` *sur l'entier* `i` *avec* `i` $= |$ `a` $|$ *et l'arbre en tas* `a`*, renvoie un arbre en tas de taille* $|$ `a` $| - 1$ *qui contient les mêmes étiquettes que* `a` *sauf celle de la racine* $\mathcal{E}$(`a`)*. Cette fonction devra au plus parcourir une fois la branche allant de la racine au dernier nœud de* `a`*. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

Question III.33 *Ecrire en CaML une fonction* `trier` *dont le type est* `int list -> int list` *telle que l'appel* `(trier l)` *sur la liste d'entiers* `l` *renvoie une liste triée d'entiers contenant les mêmes valeurs que* `l`*. Cette fonction exploitera la technique du tri par tas. Cette fonction devra être récursive ou faire appel à des fonctions auxiliaires récursives.*

Fin de l'énoncé