Arbres binaires et arbres généraux

Exercice 1.
On considère le type `type ('a,'b) abrs = Feuille of 'a | Noeud of 'b * ('a,'b) abrs * ('a,'b) abrs`.
1. Définir une fonction permettant de calculer le nombres de feuilles d'un arbre binaire de ce type.
2. Définir une fonction permettant de donner la liste des feuilles d'un arbre binaire de ce type.
3. Reprendre les deux questions précédentes pour le type `type 'a abr = Vide | Noeud of 'a* 'a abr * 'a abr;;`.

Exercice 2.
On représente les expressions arithmétiques par le type suivant :

```
type expr = Entier of int | Plus of expr*expr | Moins of expr | Mult of expr*expr | Inverse of expr ;;
```

Ecrire une fonction `valeur : expr -> float` *associant à une expression arithmétique sa valeur.*

Exercice 3.
On considère des arbres dont chaque noeud x peut avoir un nombre n (appelé arité de x) de fils. On le représente par le type

```
type 'a arbre = Vide | Noeud of 'a * ('a arbre list) ;;
```

1. Ecrire une fonction calculant la taille d'un arbre.
2. Ecrire une fonction calculant la hauteur d'un arbre.
3. Ecrire des fonctions effectuant les parcours préfixes, infixes et suffixes d'un arbre.

Exercice 4.
On appelle déséquilibre d'un arbre la différence entre la hauteur de son fils gauche et de son fils droit.
Un arbre est dit AVL (nom donné en l'honneur de Adelson, Velsky et Landis) si c'est un arbre binaire de recherche et s'il est vide ou s'il est du type $N(x, g, d)$ avec g et d des AVL et si le déséquilibre de cet arbre est -1, 0 ou 1.
1. Donner des exemples d'arbres AVL.
2. On note $T_m(h)$ la taille minimale d'un arbre AVL de hauteur h.
a. Montrer que pour tout h, $T_m(h) = T_m(h-1) + T_m(h-2) + 1$.
b. En déduire l'expression de $T_m(h)$ en fonction de h.
c. On dit qu'une famille d'arbres est équilibrée si, pour tout arbre $\mathcal{A}$ de cet famille, $h(\mathcal{A}) = O(\log(\mathrm{Card}(\mathcal{A})))$ (où $h(\mathcal{A})$ est la hauteur de l'arbre). Montrer qu'un arbre AVL est équilibré.

Exercice 5.
Un arbre de Fibonacci d'ordre p est :
-l'arbre vide si $p = 0$
-un arbre réduit à une feuille si $p = 1$
-un arbre dont le sous-arbre gauche est un arbre de Fibonacci d'ordre $p-1$ et le sous-arbre droit un arbre de Fibonacci d'ordre $p-2$.
1. Dessiner un arbre de Fibonacci d'ordre 5.
2. Déterminer le nombre de feuilles d'un arbre de Fibonacci.

Exercice 6.
Un arbre binomial d'ordre k est :
-une feuille si $k = 0$
-un arbre ayant k sous-arbres qui sont dans l'ordre des arbres binomiaux d'ordre $k-1, k-2, \cdots, 0$.
1. Donner un exemple d'arbre binomial d'ordre 4.
2. Déterminer la taille d'un arbre binomial d'ordre k.
3. Déterminer la hauteur d'un arbre binomial d'ordre k.
4. Déterminer le nombre de noeuds de profondeur p dans un arbre binomial d'ordre k.

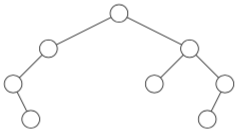
Exercice 7.
Un arbre rouge-noir est un arbre binaire de recherche dont les noeuds sont colorés en rouge ou en noir avec les propriétés suivantes :
-la racine est noire
-si un noeud est rouge, ses fils sont noirs
-pour chaque noeud, tout chemin remontant de ce noeud à la racine possède le même nombre de noeuds noirs.
1. Montrer qu'il existe des arbres binaires qui ne peuvent être coloriés en arbres rouge-noir.
2. Pour A un arbre rouge-noir, on note $b(A)$ le nombre de noeuds noirs de la racine à un feuille.
a. Montrer que $b(A) \leq h(A)+1 \leq 2b(A)$ et que $\mathrm{Card}(A) \geq 2^{b(A)}-1$.
b. En déduire que les arbres rouge-noir sont équilibrés au sens de la définition de l'exercice sur les arbres AVL.

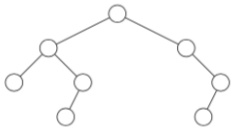
Exercice 8.
On considère dans cet exercice des arbres binaires (chaque noeud a au plus deux fils). Etant donné un tel arbre T, à tout sous-ensemble S des arêtes de T, on associe un ensemble d'arbre $S(T)$ constitué par les sommets adjacents à ces arêtes et par ces arêtes (un ensemble d'arbres est appelé une forêt).
Etant donné un arbre T, on s'intéresse à des conditions Φ et aux sous-ensembles S de T tels que $S(T)$ satisfasse la condition Φ.
1. On considère l'arbre binaire T de sommets $[\![0,11]\!]$ tel que 0 a pour fils $1,2$, 1 a pour fils $3,4$, 2 a pour fils $5,6$, 3 a pour fils $7,8$, 4 a pour fils 9, 6 a pour fils $10,11$. On considère $S=\{\{0,2\},\{1,3\},\{3,7\},\{3,8\},\{4,9\},\{6,10\},\{6,11\}\}$.
a. Dessiner cet arbre et la forêt $S(T)$.
b. Combien de sous-ensembles S' donnent la forêt $S(T)$?
c. Combien de sous-ensembles S' donnent un ensemble $S'(T)$ consistant en exactement deux arêtes isolées ?
2. On cherche à déterminer, étant donnés un arbre T et une hauteur $h \geq 1$, le nombre de sous-ensembles S de T tels que $S(T)$ contienne un arbre de hauteur au moins h. Proposer un algorithme naïf pour cette tâche et discuter sa complexité en fonction de T et de h.
3. Proposer un algorithme plus efficace pour cette tâche reposant sur la programmation dynamique. Discuter sa complexité en fonction de T et de h.
4. Comment peuton compter le nombre de sous-ensembles S tels que $S(T)$ contienne un arbre de hauteur exactement h.
5. Le diamètre d'une forêt F est le plus grand entier $d \geq 1$ tel qu'il existe un chemin (non orienté) de longueur d dans F. Proposer un algorithme qui, étant donnés T et d, calcule le nombre de sous-ensemble S de T tels que $S(T)$ soit de diamètre au moins d.

Exercice 9.
On représente en Ocaml une permutation σ de $[\![0,n-1]\!]$ par la liste d'entiers $[\sigma(0),\cdots,\sigma(n-1)]$. On utilisera le type arbre binaire suivant :

```
type arbre = V | N of arbre * int * arbre
```

On représente un arbre binaire non étiqueté par un arbre binaire étiqueté en ignorant les étiquettes. On étiquette un arbre binaire non étiqueté à n noeuds par $[\![0,n-1]\!]$ en suivant l'ordre infixe de son parcours en profondeur. La permutation associé à cet arbre est donnée par le parcours en profondeur par ordre préfixe.
1. Dessiner puis étiqueter les arbres binaires non étiquetés suivants


option/TD22-23/arbre1.png


option/TD22-23/arbre2.png

2. Ecrire une fonction `parcours_prefixe : arbre -> int list` *qui renvoie la liste des étiquettes d'un arbre dans l'ordre préfixe de son parcours en profondeur. On pourra utiliser l'opérateur de concaténation d'OCaml et on ne cherchera pas nécessairement à proposer une solution linéaire en la taille de l'arbre.*
3. Ecrire une fonction `etiquette : arbre -> arbre` *qui prend en paramètre un arbre dont on ignore les étiquettes et qui renvoie un arbre identique étiqueté par $[\![0,n-1]\!]$. On pourra utiliser une fonction auxiliaire de signature* `arbre -> int -> arbre*int` *qui prend en arguments un arbre et la prochaine étiquette à mettre et qui renvoie le couple formé de l'arbre étiqueté et la nouvelle prochaine étiquette à mettre.*
On considère l'algorithme suivant :
I - initialiser une pile vide
II - pour chaque élément en entrée faire i) tant que l'élément est plus grand que le sommet de la pile, dépiler le

sommet de la pile vers la sortie
ii) empiler l'élément en entrée dans la pile
III - Dépiler tous les éléments restants dans la pile vers la sortie

Par exemple, pour la permutation `[3;2;0;1;5;4;7;6]`*, on empile 3,2,0, on dépile 0, on empile 1, on dépile 1, 2, 3, on empile 5, 4, on dépile 4, 5, on empile 7, on empile 6, on dépile 6, 7. En ajoutant les éléments dépilés dans une liste, on obtient la liste triée* `[0;1;2;3;4;5;6;7]`*.*
4. Appliquer cet algorithme sur le deuxième arbre donné en exemple ci-dessus.
5. Ecrire une fonction `trier : int list -> int list` *qui implémente cet algorithme dans un cadre fonctionnel. On pourra utiliser le type* `list` *pour implémenter des piles. On pourra écrire une fonction auxiliaire de signature* `int list -> int list -> int list -> int list` *qui prend en arguments une liste d'entrée, une pile et une liste de sortie qui, en fonction de la forme de la liste d'entrée et de la pile, applique une étape élémentaire de l'algorithme précédent avant de procéder récursivement.*
6. Montrer que, s'il existe $0 \leq i < j < k \leq n-1$ *tels que* $\sigma(k) < \sigma(i) < \sigma(j)$*, l'algorithme précédent appliqué à la permutation* σ *ne trie pas correctement. On dit qu'une telle permutation n'est pas triable par pile.*
7. a. Montrer que la permutation associée à un arbre binaire est triable par pile.
b. Montrer qu'une permutation triable par pile est une permutation associée à un arbre binaire. On pourra prendre $\sigma(0)$ *pour racine puis procéder récursivement pour construire avec les* $\sigma(0)-1$ *premiers éléments le fils gauche et avec les autres le fils droit.*

Arbres binaires de recherche

Exercice 10.
On considère des arbres binaires de recherche étiquetés par des couples $(cl, valeur)$ *ordonnés par leurs clés.*
Ecrire les fonctions `cherche`*,* `insere`*,* `supprime` *permettant de trouver la valeur associée à une clé, d'insérer un nouveau couple clé-valeur et de supprimer le couple associé à une valeur.*

Exercice 11.
Ecrire une fonction vérifiant si un arbre binaire est un arbre binaire de recherche.

Exercice 12.
Soit u_n *le nombre d'arbres binaires de recherche dont les éléments sont* $[\![1, n]\!]$*.*
Montrer que (u_n) *vérifie* $u_1 = 1$ *et la relation de récurrence*

$$\forall n \in \mathbb{N},\ u_n = \sum_{k=0}^{n-1} u_k u_{n-1-k}.$$

Cette suite sera étudiée plus tard dans l'année, peut-être en cours de mathématiques. Il s'agit des nombres de Catalan. Vous pourrez démontrer, à l'aide de la théorie des séries entières, que $u_n = \frac{1}{n+1}\binom{2n}{n}$*.*

Exercice 13.
Ecrire des fonctions "successeur" et "prédécesseur" prenant en argument un arbre et une étiquette de celui-ci permettant de trouver dans un arbre binaire de recherche le noeud situé juste après et juste avant parmi les noeuds de l'arbre pour l'ordre sur l'ensemble des noeuds de l'arbre.

Exercice 14.
On propose l'algorithme de tri suivant : on insère les éléments d'un tableau dans un arbre binaire de recherche puis on effectue le parcours infixe sur cet arbre.
Déterminer la complexité dans le pire et dans le meilleur des cas de cet algorithme.

Exercice 15.
On considère des arbres binaires de recherche qui peuvent comporter une même étiquette en plusieurs exemplaires. Pour insérer une étiquette déjà présente dans un arbre binaire de recherche, on l'insèrera dans le sous-arbre gauche de cette étiquette. On ne cherchera pas à équilibrer les arbres.
1. Insérer successivement dans un arbre binaire de recherche initialement vide les lettres du mot "bacddabdbae" en utilisant l'ordre alphabétique. Donner la hauteur de l'arbre obtenu.
2. Montrer que le parcours en profondeur infixe d'un arbre binaire de recherche de lettres est un mot dont les lettres sont rangées dans l'ordre croissant. On pourra procéder par induction structurelle.

3. Proposer un algorithme qui permet de compter le nombre d'occurrences d'une lettre dans un arbre binaire de recherche de lettres. Quelle est sa complexité ?
4. Ecrire cet algorithme en langage Ocaml.
5. On souhaite supprimer une occurrence d'une lettre donnée d'un arbre binaire de recherche de lettres. Donner un algorithme pour ce faire et l'appliquer sur l'arbre de la question 2 en supprimant successivement une occurrences des lettres e,b,b,c et d. Donner sa complexité.
6. Ecrire cet algorithme en Ocaml.

Tas

Exercice 16.
Ecrire une fonction permettant de passer de la représentation d'un tas sous celle de tableau à celle sous forme d'arbre binaire. Définir une fonction réalisant la réciproque.

Exercice 17.
Ecrire une fonction vérifiant si un tableau d'entiers représente bien un tas.

Exercice 18.
1. Montrer que les feuilles d'un tableau de taille n représentant un tas sont les éléments d'indice supérieur à $\lfloor \frac{n}{2} \rfloor$.
2. Ecrire une fonction prenant en argument un tas-max renvoyant le plus petit élément de ce tas.

Exercice 19.
On pourra se référer à l'exercice de début de TD sur les arbres binomiaux.
On dit qu'un arbre est un arbre tournoi si chaque noeud est supérieur à chacun de ses fils. Un tas binomial est un ensemble fini d'arbre binomiaux d'ordre deux à deux distincts tels que chacun de ces arbres est un arbre tournoi.
1. Déterminer un algorithme permettant de fusionner deux arbres tournois binomiaux d'ordre k (dont les étiquettes appartiennent à un même ensemble) en un arbre tournoi binomial d'ordre $k+1$.
2. Déterminer un algorithme permettant d'insérer un nouvel arbre tournoi binomial d'ordre k dans un tas binomial.
3. Déterminer un algorithme permettant d'ajouter une étiquette à un tas binomial. L'étiquette devra appartenir à l'un des arbre du tas. La structure de tas binomial devra être préservée.

Exercice 20.
Ecrire une fonction qui prend en argument un tas représenté sous forme de tableau et qui, lorsque le tas est plein, crée un tableau représentant le même tas mais de capacité double.

Exercice 21.
On fixe un entier $d \geq 1$. *Un arbre d-aire est un arbre dont chaque noeud a au plus d fils. Un arbre de hauteur h est quasi-complet si, pour chaque profondeur* $0 \leq k \leq h-1$, *il y a* 2^k *noeuds de tel profondeur et tous les noeuds de profondeur h sont "à gauche". Un tas-max d-aire est un arbre d-aire quasi-complet tel que l'étiquette de chaque noeud est supérieure à celles de ses éventuels fils.*
1. En vous inspirant de la représentation vue en cours, proposer un moyen de représenter les tax-max d-aires à l'aide de tableaux.
2. Ecrire des fonctions OCaml d'ajout et de suppression d'un noeud.

Exercice 22. *On considère des tas d'entiers dont chaque noeud peut avoir un nombre arbitraire de fils. Chaque élément doit être supérieur à tous ses fils. On l'implémente de la façon suivante :*

```
type tas = Vide | Noeud of int * tas list ;;
```

1. Ecrire une fonction renvoyant l'élément maximal d'un tas. Donner sa complexité.
2. Pour fusionner deux tas, on insèrera le tas de racine minimale comme sous-arbre de la racine de l'autre arbre. Ecrire une fonction de fusion de deux tas de signature `fusion tas -> tas -> tas`
3. Pour insérer un élément dans un tas, on créera un tas à un seul noeud correspondant à l'élément à insérer et on fusionnera ce tas avec le tas initial. Ecrire cette fonction de signature `insere : int -> tas -> tas`
4. On va fusionner une liste de n tas, numérotés de 1 *à n, de la façon suivante : on fusionne, pour tout* $0 \leq k \leq n/2$, *le tas numéro* $2k+1$ *avec le tas numéro* $2k+2$ *(si* $2k+2 \leq n$*). On fusionne ensuite récursivement cette nouvelle liste de tas. Ecrire une fonction* `fusion2par2 : tas list -> tas` *correspondant à ce procédé.*
5. Pour supprimer l'élément maximal d'un tas, on fusionnera la liste de ses sous-arbres. Ecrire la fonction `supprime_max : tas -> tas` *qui effectue cette suppression.*
6. Discuter de la complexité des opérations précédentes.
7. Ecrire une fonction de tri par tas permettant de trier de façon décroissante une liste d'entiers en utilisant la structure précédente de tas.