TD 1: Arbres

1 Compréhension du cours

Exercice 1. *Vrai ou faux ?*. Les affirmations suivantes sont-elles vraies ou fausses ?

1. Un arbre à n nœuds est de hauteur au moins $\lfloor \log_2(n) \rfloor$.
2. Un arbre à n nœuds internes et $n+1$ feuilles est binaire entier.
3. Le plus grand élément d'un arbre binaire de recherche est à sa droite.
4. La racine d'un arbre binaire de recherche est la médiane de l'ensemble des étiquettes.
5. Le plus petit élément d'un tas-max est à la profondeur maximale du tas.
6. Le tableau `[|14; 9; 7; 8; 5; 3; 4; 6; 1|]` encode un tas-max de hauteur 3.

2 Exercices de programmation sur les arbres binaires (de recherche)

Pour les exercices qui suivent, on considère toujours une implémentation des arbres sous la forme :

```
type 'a arbre = Vide | N of 'a arbre * 'a * 'a arbre ;;
```

On ne considérera pas les noeuds `Vide` comme des vrais noeuds, les feuilles de l'arbre sont donc des noeuds dont les deux sous-arbres sont `Vide`.

Exercice 2.

1. Écrire des fonctions donnant la hauteur de l'arbre et son nombre d'éléments.
2. Écrire une fonction retournant le miroir d'un arbre (symétrie par rapport à l'axe vertical).
3. Écrire une fonction calculant la *longueur de cheminement dans un arbre*, c'est à dire la somme des profondeurs de chaque feuille. La longueur de cheminement dans un arbre « Vide » est 0.

Exercice 3. *Numérotation Sosa.* On peut numéroter les nœuds d'un arbre binaire de la façon suivante : la racine a pour numéro 1, et pour un nœud donné de numéro k, ses fils gauche et droit éventuels ont pour numéro $2k$ et $2k+1$. Écrire une fonction `numerote a` de type `'a arbre -> (int * 'a) arbre` prenant en entrée un arbre binaire et renvoyant un nouvel arbre où la numérotation a été ajoutée aux étiquettes.

Exercice 4. *Parcours d'arbre binaire.*

1. Écrire une fonction retournant la liste des étiquettes d'un arbre suivant son énumération préfixe (resp. postfixe, infixe).
2. Discuter de comment obtenir une telle énumération à l'aide d'une structure de pile, sans faire usage de récursivité.
3. Écrire une fonction retournant la liste des étiquettes d'un arbre suivant son énumération de parcours en largeur (par profondeur croissante, et les nœuds les plus à gauche en cas d'égalité). *Indication : on écrira une fonction extrayant les racines d'une liste d'arbres, et une fonction extrayant leurs sous-arbres.*
4. Discuter de comment obtenir une telle énumération à l'aide d'une structure de file, sans faire usage de récursivité.

À partir de maintenant, on suppose les étiquettes distinctes.

5. Montrer qu'un nœud a est descendant d'un nœud b dans un arbre si et seulement si b est avant a dans le parcours préfixe et après dans le parcours postfixe.
6. La liste des étiquettes d'un arbre donnée par l'énumération préfixe (resp. infixe, postfixe) suffit-elle pour reconstruire l'arbre ?
7. Montrer que la donnée de l'énumération préfixe et de l'énumération infixe permet de le faire.
8. Et la données des énumérations préfixe et postfixe ? postfixe et infixe ?

Exercice 5.

1. Écrire une fonction permettant d'obtenir un tableau contenant les nœuds d'un arbre binaire, dans l'ordre de son énumération infixe.
   — à partir de la fonction de l'exercice précédent ;
   — sans utiliser de liste (la complexité doit-être linéaire en le nombre de nœuds).
2. Que pensez-vous de la stratégie suivante pour renvoyer la copie triée d'un tableau ?
   — insérer les éléments un à un dans un arbre AVL ;
   — utiliser l'une des fonctions précédentes.

Exercice 6. Écrire une fonction `tabeau_a_arbre a` de type `'a array -> 'a arbre`, permettant de transformer un arbre binaire complet à gauche stocké sous forme de tableau en sa représentation arborescente.

Exercice 7. Où peut se trouver le prédecesseur d'un élément qui n'est pas le plus petit dans un arbre binaire de recherche ? En déduire une fonction `predecesseur a x` renvoyant le prédecesseur de x dans l'arbre binaire de recherche a si x est bien dans a et n'est pas le plus petit élément, on renverra une erreur sinon.

3 D'autres exercices sur les arbres

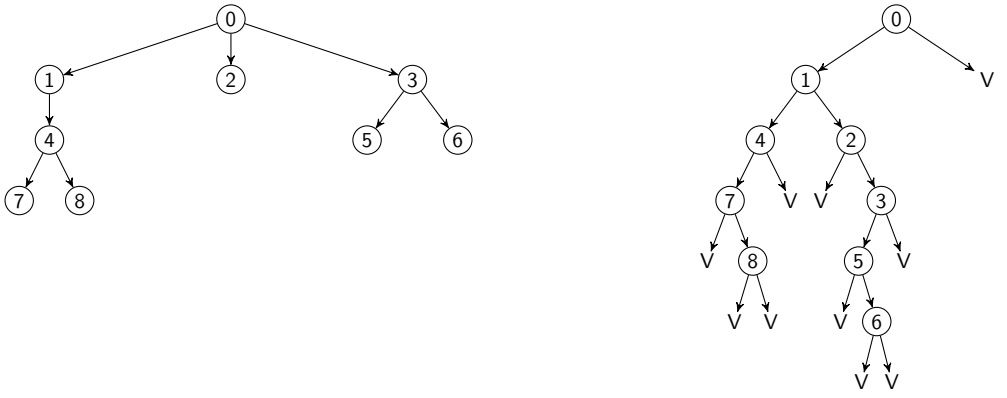
Exercice 8. Écrire des types correspondants aux arbres suivants :
— les arbres étiquetés sur les feuilles, avec un nombre arbitraire de fils ;
— les arbres binaires entiers étiquetés de manière distincte sur les feuilles, les nœuds internes, et les arêtes.

Exercice 9. On considère le type suivant, encodant un arbre étiqueté où l'arité des nœuds est arbitraire :

```
type 'a narbre = Nn of 'a * 'a narbre list ;;
```

1. Écrire une fonction renvoyant simultanément sa hauteur et son nombre d'éléments (indication : faire usage de récursivité croisée !)

Une manière d'implémenter un arbre quelconque comme un arbre binaire est d'utiliser la représentation *fils gauche, frère droit*, comme dans l'exemple ci-dessous (à gauche, un arbre donné, à droite, l'arbre binaire associé) :



La construction est la suivante. Dans l'arbre binaire, en un nœud donné x :
— le sous-arbre gauche de x correspond à la descendance de x dans l'arbre initial ;
— le sous-arbre droit de x correspond aux frères droits de x dans l'arbre initial.

En particulier, pour tout nœud y de l'arbre initial, son fils gauche dans l'arbre binaire est le même, par contre son frère droit donne son fils droit dans l'arbre binaire.

2. Écrire une fonction `rep_n_to_bin a` prenant en entrée un `narbre` et renvoyant l'`arbre` associé. Remarque : le sous-arbre droit de cet arbre est toujours Vide !
3. Inversement, à un `arbre` binaire quelconque correspond une *liste* de `narbre`. Écrire une fonction `rep_bin_to_n a` construisant la liste de `narbre` en question.

4 Exercices théoriques

Exercice 10. *Nombre d'ABR.* Donner une relation de récurrence permettant de calculer le nombre d'ABR à n étiquettes distinctes (fixées). (La solution est donné par le nombre de Catalan $C_n = \frac{1}{n+1}\binom{2n}{n}$).

Exercice 11. Déterminer le nombre d'arbres binaires de recherche de hauteur 2 (resp. 3) associés aux éléments $\{1, 2, 3, 4, 5, 6\}$.

Exercice 12. *Tas fusionnables.* Les tas (max) supportent les opérations d'ajout d'un élément et de suppression du maximum. Fusionner deux tas prend *a priori* un temps linéaire en le nombre d'éléments dans le pire cas : il faut reconstruire un tas à partir des éléments des deux tas (et il est possible de construire un tas en temps linéaire en le nombre d'éléments). On décrit ici une structure de tas permettant en plus de réaliser la fusion, de manière effiace. On appelle *arbre binomial*, un arbre dont la structure est définie comme suit :

— l'arbre B_0 possède un seul noeud ;

— l'arbre B_k pour $k \geq 1$ a sa racine d'arité k, son i-ème fils étant la racine d'un arbre B_i, pour $0 \leq i < k$.

1. Dessiner B_1, B_2, B_3.

2. Donner le nombre de noeuds de l'arbre B_i.

3. Expliquer comme former un arbre B_{i+1} à partir de deux arbres B_i.

4. Donner le nombre de noeuds de l'arbre B_i se trouvant à profondeur p.

On définit un *tas binomial (max)* comme un ensemble d'arbres binomiaux B_i de tailles *distinctes*, tel que chacun des arbres binomiaux possède la structure de tas (max) : l'étiquette d'un nœud est plus grande que celle de ses enfants.

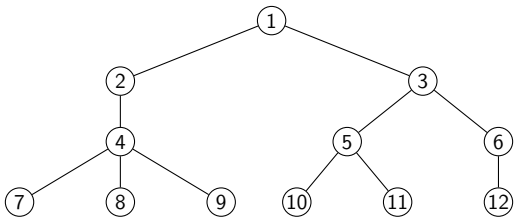
5. Justifier que pour tout $n \geq 0$, il n'y a qu'une seule façon de décomposer n en somme de tailles distinctes d'arbres binomiaux.

6. Expliquer comment déterminer le maximum d'un tas binomial.

7. Expliquer comment faire l'union de deux tas binomiaux stockés comme des listes d'arbres binomiaux de taille croissante de façon efficace. On suppose que l'on peut construire un arbre binomial B_{i+1} à partir de deux arbres B_i et réciproquement en temps $O(1)$. Quelle est la complexité de la fusion de deux tas binomiaux comportant n éléments en tout ?

8. Estimer la complexité de l'insertion d'un nouvel élément/la suppression du maximum dans un tas binomial.

Exercice 13. *Oral ENS.* Soit un arbre enraciné infini, tel que chaque sommet admet un nombre fini non nul de fils. On numérote ses sommets dans un parcours en largeur « de gauche à droite », la racine étant indexée par 1, comme dans l'exemple ci-dessous (on n'a représenté que les 4 premiers niveaux).



La fonction de parenté de l'arbre est la fonction de $\mathbb{N}$ vers $\mathbb{N}$ qui envoie 0 sur 0, 1 sur 1, et chaque entier $i > 1$ vers le numéro du père du sommet de numéro i. Dans l'exemple ci-dessus, la fonction envoie 5 sur 3 et 9 sur 4.

1. Soit $f : \mathbb{N} \to \mathbb{N}$. Montrer que

— f surjective ;

— f croissante ;

— $f(2) = 1$.

est une condition nécessaire et suffisante pour qu'une fonction de $\mathbb{N}$ dans $\mathbb{N}$ soit une fonction de parenté.

2. Soit f la fonction définie par $f(0) = 0$ et $\forall n > 0, \quad f(n) = n - f(n-1)$. Montrer que f est une fonction de parenté et dessiner les premières couches de l'arbre associé.

3. Montrer qu'on définit bien une fonction de $\mathbb{N}$ vers $\mathbb{N}$ en posant $g(0) = 0$ et

$$\forall n > 0, \qquad g(n) = n - g(g(n-1)).$$

4. Montrer que g est une fonction de parenté et dessiner les premières couches de l'arbre associé. *Indication : montrer que* $g(n) - g(n-1) \in \{0, 1\}$ *pour tout* $n \geq 1$.

5. Montrer que le plus grand fils de k a pour étiquette $k + g(k)$, pour tout $k \geq 1$.

6. Soit $(F_i)_{i\geq 0}$ la suite de Fibonacci vérifiant $F_0 = F_1 = 1$ et $F_i = F_{i-1} + F_{i-2}$ pour $i \geq 2$. Calculer les valeurs de $(g(F_i))_i$.

7. Montrer que tout entier n s'écrit de façon unique comme somme de termes de la suite de Fibonacci dont les indices sont non nuls, deux à deux distincts et non consécutifs.

8. Exprimer g en fonction de cette décomposition.