Feuille de révisions n°3 - Programmation en OCaml : manipulation d'arbres

Dans toute cette feuille de révision on suppose définis les types OCaml suivants, permettant respectivement la représentation d'arbres binaires non vides, d'arbres généraux non vides.

```
type 'a btree =
  | E                              (* arbre vide *)
  | N of 'a * 'a btree * 'a btree (* noeud interne *)

type 'a gtree =
  | GN of 'a * 'a gtree list
```

1 Pied à l'étrier

R. 3-1 Définir une fonction permettant de tester si un arbre binaire est vide.

R. 3-2 Définir une fonction calculant la hauteur d'un arbre binaire. On rappelle que la hauteur de l'arbre vide est -1.

R. 3-3 Définir une fonction calculant la taille (*i.e.* le nombre de nœuds) d'un arbre général. On itérera sur l'arbre général au moyen de deux fonctions mutuellement récursives.

R. 3-4 Définir une fonction calculant la taille *i.e.* (le nombre de nœuds) d'un arbre général. On itérera dans l'arbre général au moyen d'une unique fonction récursive et d'un `List.fold_left`.

R. 3-5 Définir une fonction calculant la taille (*i.e.* le nombre de nœuds) d'un arbre général. On fournira une implémentation récursive terminale.

R. 3-6 Définir une fonction permettant de tester l'appartenance d'une étiquette à un arbre binaire.

R. 3-7 Définir une fonction prenant en argument un arbre binaire étiqueté par des entiers et retournant `None` si l'arbre est vide et `Some(mi, ma)` où `mi` est l'étiquette minimale de l'arbre et `ma` l'étiquette maximale sinon. Cette fonction sera récursive et ne devra pas faire usage d'une fonction récursive auxiliaire.

R. 3-8 Définir une fonction prenant en argument un arbre binaire étiqueté par des entiers et retournant `None` si l'arbre est vide et `Some(mi, ma)` où `mi` est l'étiquette minimale de l'arbre et `ma` l'étiquette maximale sinon. Cette fonction ne sera pas récursive, elle devra faire appel à une fonction récursive auxiliaire qui sera récursive terminale.

R. 3-9 Définir une fonction prenant en arguments deux arbres binaires et testant si ceux-ci ont la même forme, c'est-à-dire si les deux arbres sont égaux lorsqu'on omet les valeurs des étiquettes.

R. 3-10 Définir une fonction prenant en argument un arbre binaire étiqueté par des entiers et retournant `Some(p)` où `p` est la profondeur d'un nœud d'étiquette paire s'il en existe et `None` sinon. On s'efforcera d'interrompre la recherche dès que cela est possible, en utilisant un mécanisme d'exception.

2 Parcours d'arbres

R. 3-11 Définir une fonction calculant le parcours infixe d'un arbre binaire. On pourra fournir une implémentation de complexité quadratique.

R. 3-12 Définir une fonction calculant le parcours infixe d'un arbre binaire. On fournira une implémentation de complexité linéaire.

R. 3-13 Définir une fonction prenant en argument un arbre binaire a et calculant un arbre binaire dont le parcours infixe est le miroir du parcours infixe de a.

R. 3-14 Définir une fonction calculant la bijection vue en premiere année entre l'ensemble des listes d'arbres généralisés (`'a gtree list`) et les arbres binaires (`'a btree`). Définir ensuite sa réciproque.

R. 3-15 Définir une fonction retournant un parcours en profondeur d'un arbre binaire. On doit fournir une implémentation récursive n'utilisant pas de fonction récursive auxiliaire. Cette implémentation peut être non récursive terminale, de complexité quadratique.

R. 3-16 Définir une fonction retournant un parcours en profondeur d'un arbre binaire. On doit fournir une implémentation récursive terminale de complexité linéaire.

R. 3-17 Définir une fonction retournant un parcours en profondeur d'un arbre binaire. On doit fournir une implémentation récursive terminale de complexité linéaire en utilisant le module `Stack`.

R. 3-18 Définir une fonction retournant un parcours en largeur d'un arbre binaire (la liste de ses étiquettes, triées par profondeur, et à profondeur équivalente de gauche à droite). On doit fournir une implémentation récursive terminale de complexité linéaire en utilisant le module `Queue`.

R. 3-19 Définir une fonction retournant un parcours en largeur d'un arbre binaire. On doit fournir une implémentation récursive terminale de complexité linéaire en utilisant deux listes : celle du niveau courant, celle du niveau suivant.

3 Utilisations classiques en algorithmique

R. 3-20 Définir une fonction permettant de tester l'appartenance d'un élément dans un arbre binaire de recherche. On fournira une implémentation ayant une complexité linéaire en la hauteur de l'arbre considéré.

R. 3-21 Définir une fonction prenant en argument un arbre binaire étiqueté par des entiers et testant si l'arbre est de recherche, à savoir si pour tout nœud de l'arbre d'étiquette e, toute étiquette de son fils gauche est inférieure stricte à e, et toute étiquette de son fils droit est supérieur à e. On fournira une implémentation ayant une complexité en $\mathcal{O}(n^2)$.

R. 3-22 Définir une fonction prenant en argument un arbre binaire étiqueté par des entiers et testant si l'arbre est de recherche, à savoir si pour tout nœud de l'arbre d'étiquette e, toute étiquette de son fils gauche est inférieure à e, et toute étiquette de son fils droit est supérieur à e. On fournira une implémentation ayant une complexité en $\mathcal{O}(n)$.

R. 3-23 Définir une fonction prenant en argument un arbre binaire et testant si celui-ci est parfait c'est à dire que tout nœud a soit deux fils vides (autrement dit est une feuille), soit deux fils non vides et que toutes les feuilles de l'arbre ont même profondeur.

R. 3-24 Définir une fonction prenant en argument un arbre binaire et testant si celui-ci est parfait c'est à dire que tout nœud a soit deux fils vides (autrement dit est une feuille), soit deux fils non vides et que toutes les feuilles de l'arbre ont même profondeur.
On fournira une implémentation ayant une complexité en $\mathcal{O}(n)$.

R. 3-25 Implémenter une fonction de tri par arbre binaire de recherche : pour trier une liste d'éléments, on insère successivement ces éléments dans un arbre binaire de recherche, un parcours infixe de l'arbre binaire de recherche fourni alors un tri de la liste initiale.

R. 3-26 Implémenter une fonction de tri par tas : pour trier une liste d'éléments, on insère successivement ces éléments dans une file de priorité implémentée par un tas binaire complet, des extractions successives du minimum permettent alors d'obtenir un tri de la liste initiale.