Travaux Pratiques d'AP3 n°5

—Licence Informatique, 3ième année, 1ier semestre—

Implantation et usage des arbres binaires de recherche

Le but de ce TP est d'implanter un module pour les arbres binaires de recherche et de l'utiliser.

▸ **Exercice 1. (Un module pour les arbres binaires de recherche)** *L'objectif de cet exercice est d'implanter un module pour les arbres binaires de recherche (ABR). Pour cela commencez par ouvrir un fichier nommé* `bst.ml` *(« bst » pour binary search tree).*

1. *Dans le fichier* `bst.ml` *en procédant comme d'habitude et à partir des algorithmes vus en cours, vous écrivez le code des fonctions* `bst_seek` *qui recherche un élément dans un ABR,* `bst_linsert` *qui insère aux feuilles un élément dans un ABR,* `bst_lbuild` *qui construit un ABR à partir d'une liste d'éléments avec des insertions aux feuilles,* `bst_delete` *qui supprime un élément dans un ABR.*
2. *Maintenant vous créez un fichier* `bst_exercise.ml` *où vous allez recopier tous les exemples et essais que vous avez fait lors de l'écriture des fonctions précédentes et vous supprimez ces mêmes exemples et essais du fichier* `bst.ml`*.*
3. *Vous compilez le fichier* `bst.ml` *l'aide de la commande* `ocamlc -c bst.ml` *afin d'obtenir un fichier* `bst.cmo`*. Comme dans* `bst.ml` *vous utilisez les arbres binaires il doit y avoir une instruction* `open Btree` *dans le fichier* `bst.ml`*.*
4. *Dans le fichier* `bst_exercise.ml` *vous ajoutez les commandes* `#load` *et les instructions* `open` *afin de pouvoir exécuter vos exemples et essais sur les ABR. Normalement, vous avez dû créer un module* `Bst` *dont vous pouvez voir le contenu avec la commande* `#show Bst`*.*
5. *Dans le fichier* `bst_exercise`*, écrivez une fonction* `bst_isBst` *qui teste si un arbre est un ABR.*

▸ **Exercice 2.** *Pour l'algorithme d'ajout à la racine dans un arbre binaire de recherche (ABR), on a besoin de couper en deux l'arbre a où l'on veut insérer en fonction de l'élément e que l'on veut insérer. Dans le cours, nous avons vu la spécification de cet algorithme. L'objectif de cet exercice est d'en réaliser une implantation.*

1. *Construisez l'ABR correspondant à la suite d'insertions aux feuilles q, d, a, i, e, g, l, t.*

2. *Coupez l'arbre obtenu selon le caractère f pour cela procédez par étape afin de découvrir un algorithme de coupe.*
3. *En vous aidant de la spécification de l'opération* `ajt_r` *vue en cours et dans le fichier* `bst.ml`*, ajoutez le code d'une fonction* `bst_cut` *qui réalise la coupe d'un ABR selon un élément.*
4. *Montrez par récurrence sur la hauteur des arbres que l'algorithme de coupure construit un ABR g qui contient tous les éléments de a qui sont inférieurs ou égaux à e et un arbre d qui contient tous les éléments de a qui sont strictement supérieurs à e.*
5. *Dans le fichier* `bst.ml`*, ajoutez le code le code d'une fonction* `bst_rinsert` *et vérifiez que les axiomes de l'opération* `ajt_r` *sont satisfaits.*

▸ **Exercice 3. (File de priorité avec un ABR)** *Une file de priorité est une file d'éléments munis d'une valeur qui représente la priorité d'un élément. La tête de la file est l'élément ayant la plus grande (ou la plus petite) priorité. L'utilisation d'une liste pour implanter une file de priorité n'est pas une implantation très efficace. Une idée plus intéressante est d'utiliser un arbre binaire de recherche.*

1. *Utilisez votre module sur les ABR pour réaliser un module* `pqueue` *qui implante les files de priorité avec les ABR. (Indication : vous devez modifier votre module sur les ABR afin de pouvoir donner la fonction de comparaison). Votre implantation doit permettre de choisir entre une file de priorité dont les éléments sont triés selon les priorités croissantes ou une file de priorité dont les éléments sont triés selon les priorités décroissantes.*
2. *Cette implantation est plus efficace que l'utilisation d'une liste, mais elle souffre de quelques défauts du point de vue de l'efficacité, sauriez-vous les décrire ?*