Auteur : R. Namyst
Enseignants : L. Bouge, A. Bonhomme
Email : {Lub.Bouge,Alice.Bonhomme}@ens-lyon.fr

le 09/10/00

TD Programmation 2 : Arbres Binaires

1 Définition et manipulation d'un arbre binaire

1.1 définition

Voici la définition du type `itree` que vous utiliserez :

```
type itree = Empty | Node of node
and node = {item: int; left: itree; right: itree}
;;
```

1.2 Fonctions élémentaires : constructeurs, extracteurs

Munissez le type `itree` de quelques opérations élémentaires de manière à pouvoir le manipuler comme un type abstrait :

`empty : unit -> itree`
Retourne un arbre vide.
`make : int -> itree -> itree -> itree`
Construit un arbre à partir d'un élément (un entier), d'un fils droit et d'un fils gauche.
`get_item : itree -> int`
Retourne l'élément en tête de liste.
`get_right itree -> itree`
Retourne le fils droit de l'arbre.
`get_left itree -> itree`
Retourne le fils gauche de l'arbre.
`is_empty itree -> bool`
Indique si l'arbre est vide.

Important ! *À partir de maintenant, les arbres doivent être* exclusivement *manipulés à travers ces fonctions.*

1.3 Fonctions simples : observateurs

En utilisant les opérations élémentaires précédentes, définissez les fonctions suivantes :

`size : itree -> int`
Retourne le nombre d'éléments dans l'arbre.
`height: itree -> int`
Retourne la hauteur de l'arbre
`is_equal: itree -> itree -> bool`
Indique si les arbres sont semblables.

1.4 Génération aléatoire d'un arbre

Nous voulons écrire une fonction `generate_tree n k` qui génère aléatoirement un arbre tel que :

- k est la valeur maximale que peut prendre un élément (*item*) de l'arbre généré,
- n est la taille de l'arbre.

Pour ceci nous proposons les règles suivantes de construction d'un arbre aléatoire :

- choisir aléatoirement la valeur de l'élément entre 0 et k,
- choisir aléatoirement une valeur p entre 0 et n,
- Générer aléatoirement un arbre gauche de hauteur maximum p,
- Générer aléatoirement un arbre droit de hauteur maximum $n - p - 1$.

Utiliser la fonction `Randon.int` (`Random.int bound` returns a random integer between 0 (inclusive) and `bound` (exclusive). `bound` must be more than 0 and less than 230.)

1.5 Parcours d'arbre

A l'aide des fonctions élémentaires prédéfinies, écrire les fonction suivantes :

`prefix_print_tree: itree -> unit`
Imprime les éléments de l'arbre d'après un parcours préfixe
`infix_print_tree: itree -> unit`
Imprime les éléments de l'arbre d'après un parcours infixe
`postfix_print_tree: itree -> unit`
Imprime les éléments de l'arbre d'après un parcours postfixe
`flatten: itree -> int list`
Aplatit l'arbre en liste d'après un parcours préfixe
`copy: itree -> itree`
Retourne une copie de l'arbre.
`increment:  itree -> itree`
Retourne une copie de l'arbre dont les valeurs ont été incrémentées de 1.
`sum:  itree -> int`
Retourne la somme de tous les éléments de l'arbre.

2 Arbres binaires ordonnés

Il s'agit de manipuler des arbres binaires ordonnés contenant des entiers, c'est-à-dire des arbres où pour tout nœud n de l'arbre, les valeurs des nœuds situés dans le fils gauche (resp. droit) sont toutes $\leq$ (resp. $\geq$) à la valeur du nœud n.

La figure 1 donne un exemple d'arbre binaire ordonné d'entiers.

A l'aide des fonctions élémentaires prédéfinies, écrire les fonction suivantes :

`is_sorted:  itree -> bool`
Vérifie si un arbre est ordonné.
`search:  int -> itree -> bool`
Vérifie si un élément est présent ou non dans un un arbre trié.
`insert:  int -> itree -> itree`
Génère un arbre trié résultat de l'insertion d'un nouvel élément dans un arbre trié.
`sort:  itree -> itree`
Retourne l'arbre trié.

Pour afficher le contenu d'un arbre trié (en respectant l'ordre), quel parcours d'arbre doit-on utiliser ?

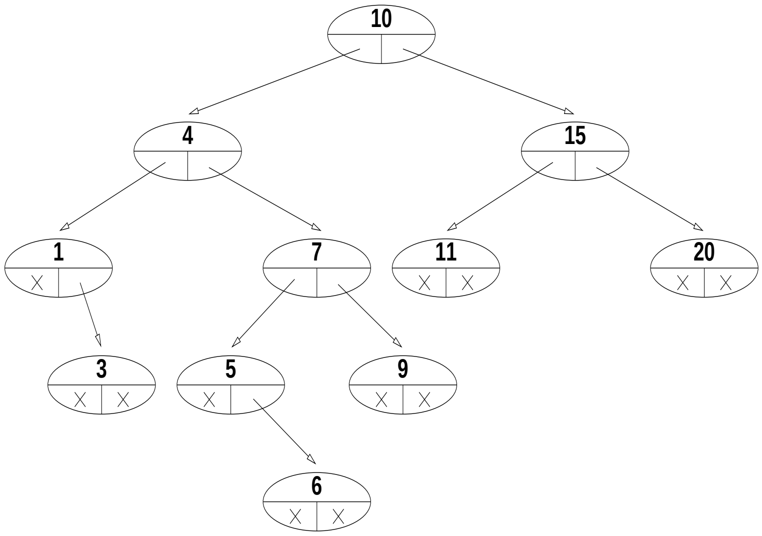

FIG. 1: Arbre binaire ordonné contenant 11 entiers

3 Une autre manière de parcourir les arbres

On définit le type `itree_bag` (appelé multi-ensemble) suivant :

```
type itree_bag = itree list
```

3.1 Fonctions élémentaires

Ecrire les fonctions :

`empty_bag  unit -> itree_bag`
Retourne un multi-ensemble `itree_bag` vide

`is_empty_bag   itree_bag -> bool`
Vérifie si un multi-ensemble est vide

`add_bag  itree -> itree_bag -> itree_bag`
Ajoute un arbre dans le multi-ensemble.

`remove_bag  itree_bag -> itree * itree_bag`
Retourne le dernier élément inséré dans le multi-ensemble et le sous-ensemble privé de cet élément.

3.2 Parcours d'un arbre à l'aide des `itree_bag`

A partir des fonctions précédemment définies, écrire une fonction permettant de parcourir un arbre dans un ordre préfixé, puis dans un ordre postfixé.

Changer la fonction `remove_bag` pour que ce parcours soit un parcours en largeur.

4 Itérateurs d'arbres

En vous inspirant des fonctions de la section 1.5, écrire les itérateurs `map, reduce, do` pour les arbres