TRAVAUX PRATIQUES XVI

Arbres Binaires de Recherche (ABR)

Le but de ce TP est de programmer les opérations vues en cours sur les arbres binaires de recherche. L'idée est de :

1) commencer par essayer d'écrire la fonction sans consulter le cours
2) au besoin, consulter la description de l'algorithme dans le cours
3) en désespoir de cause, consulter le pseudo-code

On travaille en OCaml, et nous ferons donc une structure de donnée fonctionnelle. Rappelons que cela signifie que nos fonctions ne vont pas *modifier* leur argument, mais renvoyer un *nouvel* arbre qui est le résultat de l'opération.

La récursivité sera indispensable. On rappelle que lorsque l'énoncé (ou un code OCaml écrit `f : type`), cela signifie que `f` a pour type le type indiqué. Par exemple, `f : int -> int -> bool` est une fonction qui prend en argument deux entiers et renvoie un booléen.

Il est fortement recommandé pour ce TP de *faire des dessins !*.

A Opérations sur les ABR

On utilisera le type suivant pour les ABR :

OCaml
```
type 'a abr = Nil | N of 'a abr * 'a * 'a abr
```

1. Écrire un abr donné en exemple en cours dans ce type. Il nous servira à faire des tests.

Recherche dans un ABR

2. Écrire une fonction `mem_arbre : 'a -> 'a abr -> bool` testant si une certaine clé est présente dans un arbre. Testez.

A.1 Insertion dans un ABR

3. Écrire une fonction `insere_arbre : 'a -> 'a abr -> 'a abr` qui insère un élèment dans un arbre. On travaillera *sans doublons* : si l'élément est déjà présent, votre fonction doit renvoyer l'arbre initial inchangé. Testez.

4. (Bonus) En déduire une fonction qui prend en argument une liste et renvoie un abr contenant les éléments de la liste.

A.2 Tri

5. Écrire une fonction `ordonne_abr : 'a abr -> 'a list` qui prend en argument un abr et renvoie la liste de ses éléments, triée.

Si jamais vous obtenez la liste « à l'envers», vous pouvez utiliser `List.rev` sur le résultat pour le renverser et remettre la liste « à l'endroit ».

6. (Bonus) Écrire une fonction qui convertit une liste en abr. En déduire une fonction qui trie une liste à l'aide d'un abr.

A.3 Suppression dans un ABR

On veut implémenter la suppression efficace du cours.

7. Écrire une fonction `min_arbre : 'a abr -> 'a` qui renvoie le minimum d'un abr.

8. Écrire une fonction `supprime_max : 'a abr -> 'a abr` qui supprime le maximum du sous-arbre passé en argument.

9. Écrire unefonction `supprime_element : 'a -> 'a abr -> 'a abr` qui supprime un noeud d'un arbre. Si le noeud n'est pas dans l'arbre, la fonction ne fait rien.

B Pour les plus rapides

10. Cherchez comment générer une liste aléatoire. Étudiez l'efficacité en pratique du tri par abr. Vous pouvez utiliser en terminal la commande `time` pour mesurer le temps d'exécution d'un programme.

    Il pourra être pertinent de comparer ce cas aléatoire au meilleur et au pire cas expérimentaux, ainsi qu'à `List.fast_sort` .
11. Une fois tout cela terminé, débloquez le lvl 4 de F-IOI. Puis le 5.