Arbres binaires de recherche

Résumé des aventures précédentes : on souhaite implémenter la structure de données *dictionnaire* (ou *tableau associatif*). On va voir dans ce chapitre qu'on peut utiliser des arbres binaires de recherche, mais :

- les arbres binaires de recherche ont un intêrêt en soi, indépendamment des tableaux associatifs ;
- les tableaux associatifs peuvent être implémentés autrement (cf TP04).

I Définition et intérêt des ABR

I.1 Arbre binaire de recherche

On suppose avoir fixé un ensemble d'étiquettes $\mathcal{E}$ et une application cle $: \mathcal{E} \to \mathcal{K}$ où $\mathcal{K}$ est un ensemble totalement ordonné. Autrement dit, à toute étiquette on peut associer sa **clé** et l'ensemble des clés est totalement ordonné.

Exemples 1 :

1. Si le type des étiquettes est totalement ordonné, on peut simplement prendre pour clés les étiquettes elles-mêmes.
2. Le type des étiquettes peut être de la forme `'a*'b` et les clés les éléments de type `'a` (les élément de type `'b` étant les *valeurs* des étiquettes).
3. Le type des étiquettes peut-être un type enregistrement dont l'un des champs correspond aux clés.

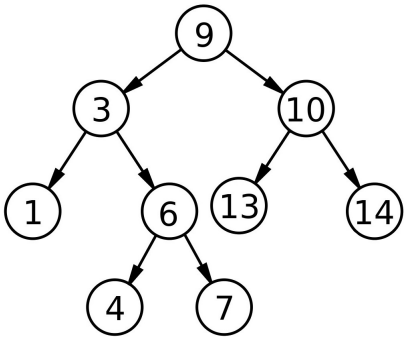
Définition 1 : Arbre binaire de recherche (ABR)

Étant donné un ensemble d'étiquettes $\mathcal{E}$ muni de clés, l'ensemble des arbres binaires de recherche (en abrégé : ABR) étiquettés par $\mathcal{E}$ est le plus petit sous-ensemble de l'ensemble des arbres binaires étiquettés par $\mathcal{E}$ tel que :

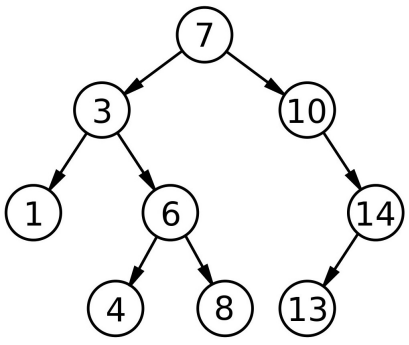
- l'arbre vide est un ABR ;
- un nœud n de fils gauche fg et de fils droit fd est un ABR si et seulement si
  - fg est un ABR ;
  - fd est un ABR ;
  - toutes les étiquettes de fg ont une clé inférieure à la clé de l'étiquette de n ;
  - toutes les étiquettes de fd ont une clé supérieure à la clé de l'étiquette de n.

Remarque 1 : Dans le cas où plusieurs nœuds peuvent avoir des étiquettes ayant la même clé, on peut retrouver la même clé à gauche ou à droite du nœud où elle apparaît pour la première fois. On peut proposer des variantes en remplaçant "inférieure" par "strictement inférieure" ou en remplaçant "supérieure" par "strictement supérieure".

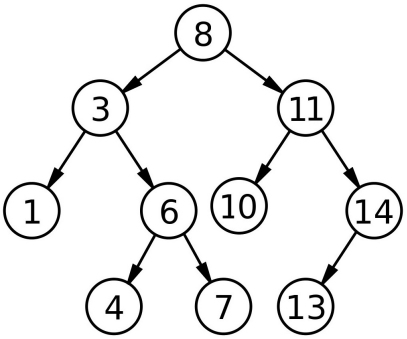
Exemples 2 : Pour faire simple, prenons l'exemple où le type des étiquettes est `int` et les clés sont les valeurs des étiquettes elle-mêmes. Les arbres suivants sont-ils des ABR ? Pourquoi ?



Arbre a1



Arbre a2



Arbre a3

I.2 Recherche dans un ABR

Commençons à voir l'intérêt des ABR en se posant la question suivante : étant donné un *arbre binaire*, comment peut-on déterminer si une étiquette de clé donnée figure dans l'arbre ? Et si l'arbre est un *ABR* ?

Par exemple : comment déterminer si 8 est dans `a1`, `a2`, `a3` (à part : "ça se voit") ? Quel intérêt des ABR pour la recherche ?

I.3 Insertion dans un ABR

Donnons l'idée de l'implémentation des dictionnaires que l'on détaillera plus bas et en TP. Imaginons que l'on souhaite implémenter le dictionnaire {"one" :"un", "two" :"deux", "three" :"trois", "four" :"quatre", "five" :"cinq", "six" :"six", "seven" :"sept", "eight" :"huit", "nine" :"nœuf", "ten" :"dix"}. Sur cet exemple, les clés sont des chaînes de caractères correspondant à des nombres en anglais, et les valeurs des chaînes de caractères correspondant à leur traduction en français. Bien sûr, ni les clés ni les valeurs d'un dictionnaire ne sont nécessairement des chaînes de caractères, pour un autre exemple cela pourrait être autre chose.

On implémente ce dictionnaire par un `string*string arbre_binaire`, où les premières coordonnées des étiquettes correspondent aux clés, et les secondes aux valeurs. Mais, pour que la recherche d'un élément dans le dictionnaire soit efficace, on décide que cet arbre binaire sera un ABR. Proposons un ABR correspondant à ce dictionnaire, en indiquant les étapes de la construction de celui-ci (l'ordre sur les clé est l'ordre lexicographique).

On devrait obtenir un ABR relativement équilibré : lorsqu'on insère des éléments sans ordre particulier les uns après les autres, c'est statistiquement ce qui se produit. On est ravi parce que cela implique que .....

Il arrive qu'on obtienne des ABR non équilibrés : on peut alors les équilibrer au cours de la construction.
Je ne suis pas censé en parler mais les plus rapides pourront voir ceci en TP.

II Implémentation

Cette partie est davantage de l'ordre des travaux pratiques, on pourra en reprendre le code (ou la traiter) lors du TP06.

On implémente les ABR par des `('a*'b) arbre_binaire`, où les clés sont les éléments de type `'a` et les valeurs les éléments de type `'b`. (Adapter aux autres variantes proposées dans l'exemple 1 ne devrait pas poser de problème.)

II.1 Vérification

On souhaite écrire une fonction `est_abr : ('a*'b) arbre_binaire → bool` permettant de tester si un arbre binaire est ou pas un ABR.

Ce n'est pas si facile : si l'arbre est non vide on doit récursivement appliquer la fonction sur le fils gauche et sur le fils droit, mais également s'assurer que toutes les clés du fils gauche (respectivement du fils droit) sont bien plus petites (respectivement plus grandes) que celle de la racine, de préférence en explorant une seule fois chaque nœud.

II.2 Recherche

Plus facile : écrivons une fonction `rechercher : 'a → ('a*'b) arbre_binaire → 'b` qui à une clé et un arbre binaire **dont on suppose que c'est un ABR**, associe la valeur du premier nœud de l'ABR ayant pour etiquette cette clé (en provoquant une exception s'il n'y en a pas). **Évidemment, il faut exploiter que l'arbre considéré est un ABR.**

II.3 Insertion

Écrivons une fonction `inserer : 'a → 'b → ('a*'b) arbre_binaire → ('a*'b) arbre_binaire` qui prend comme arguments une clé, une valeur et un ABR, et retourne l'ABR obtenu en rajoutant le couple (clé,enregistrement) donné. Si la clé apparaît déjà dans l'ABR, il y a plusieurs solutions possibles et vous pouvez choisir votre préférée.

II.4 Suppression

On souhaite écrire une fonction `supprimer : 'a → ('a*'b) arbre_binaire → ('a*'b) arbre_binaire` qui à une clé et un ABR associe l'ABR obtenu en retirant la première étiquette où apparaît la clé (si elle existe).

Avec la même méthode que pour la recherche, on peut identifier sans difficulté le nœud à supprimer (s'il existe). On est donc ramené à l'écriture d'une fonction `supprimer_racine : ('a*'b) arbre_binaire → ('a*'b) arbre_binaire` qui à un ABR non vide associe l'ABR obtenu en retirant sa racine.

Mais ceci est plus difficile !
• Supprimer une feuille ne pose pas de problème, on la remplace par l'arbre vide et c'est réglé.
• Supprimer un nœud qui n'a qu'un seul fils ne pose pas de problème non plus, on le remplace par son fils.
• Sinon, on procède comme suit : on permute le nœud avec n'importe lequel de ses fils (choisissez) et on supprime récursivement la racine de ce fils.
Go !

Et voilou.