Dictionnaires – Arbres binaires de recherche – Arbres AVL

On rappelle qu'un dictionnaire est une structure de données permettant de stocker des éléments en les repérant par des clés. Lors de l'exécution d'un programme, il est pertinent d'utiliser un dictionnaire pour stocker les variables. En effet, le programme a besoin d'accéder à la valeur d'une variable à partir de son nom. Dans ce cas, les clés du dictionnaire sont des chaînes de caractères (ce sont les noms des variables) et ses éléments sont des entiers (ce sont les valeurs des variables que l'on supposera être des entiers).

```
let ma_variable2 = 16 and
    ma_variable = 15 and
    x0 = 0 and
    x = 1 and
    y = 0 and
    a = -1 and
    b = 0;;
```

Par exemple, si on exécute le programme ci-dessus, alors les variables sont stockées en machine à l'aide d'un dictionnaire qui comporte sept clés : la valeur correspondant à la clé `"ma_variable2"` est `16`, la valeur correspondant à la clé `"ma_variable"` est `15`, et ainsi de suite ...

1 Arbres binaires de recherche

1.1 Préliminaires

Afin d'implémenter la structure de dictionnaire, on propose d'utiliser des arbres binaires de recherche. On manipulera donc des arbres binaires dont les nœuds sont étiquetés par des couples (clé, valeur) et tels que pour tout nœud x de l'arbre :

- La clé de x est supérieure à toutes les clés présentes dans le sous-arbre gauche.
- La clé de x est inférieure à toutes les clés présentes dans le sous-arbre droit.

Le dictionnaire de l'exemple précédent est représenté par les arbres des figures 1, 2, 3 et 5 qui se trouvent à la fin de l'énoncé. On remarque en particulier que plusieurs arbres binaires de recherche peuvent représenter le même dictionnaire.

Dans ce sujet, on suppose que les clés présentes dans un arbre binaire de recherche sont toutes différentes.

Mis à part dans la question 2, vous pouvez utiliser les opérateurs `<`, `<=`, `=`, `>=` et `>=` d'OCaml pour comparer deux chaînes de caractères suivant l'ordre lexicographique. Par exemple :

Expression	`"a"<"y"`	`"x"<"b"`	`"x0"<"x"`	`"ma_variable"<"ma_variable2"`	`"b"<"b"`
Valeur	`true`	`false`	`false`	`true`	`false`

On définit le type `dict` suivant pour représenter les dictionnaires :

```
type etiquette = {
    cle: string;
    valeur: int;
  };;
```

```
type dict =
  | Vide
  | N of etiquette * dict * dict;;
```

1. (a) Rappeler la définition d'une structure de données persistante.
   (b) Le type `dict` correspond-il à une structure de données persistante ?
2. Écrire une fonction « `lt: string -> string -> bool` » qui prend en entrée deux chaînes des caractères `s1` et `s2`, qui renvoie `true` si `s1 < s2` et `false` sinon. Vous pouvez utiliser les opérateurs `<`, `<=`, `=`, `>=` et `>=` pour comparer deux caractères, mais pas pour comparer deux chaînes de caractères. Pour information, le nom de la fonction `lt` correspond aux initiales de "less than".

1.2 Implémentation des opérations élémentaires sur les dictionnaires

Le but de cette partie est d'implémenter les fonctions nécessaires pour manipuler des dictionnaires, vous devez donc adapter les fonctions vues en cours.

3. Un dictionnaire vide est représenté par un arbre vide.
   (a) Écrire une fonction « `create: unit -> dict` » qui créé un dictionnaire vide.
   (b) À l'aide d'un filtrage, écrire une fonction « `is_empty: dict -> bool` » qui teste si un dictionnaire est vide.
4. Écrire une fonction « `find: dict -> string -> int` » qui prend en entrée une clé et renvoie la valeur associée à cette clé. Votre fonction déclenchera une exception si la clé n'apparaît pas dans le dictionnaire.
5. Écrire une fonction « `add: dict -> string -> int -> dict` » qui prend en entrée un dictionnaire `abr`, une clé `c` ainsi qu'une valeur `v`, et ajoute à `abr` l'association entre la clé `c` et la valeur `v`. Votre fonction déclenchera une exception si la clé apparaît déjà dans le dictionnaire.
6. Quelles sont les complexités des fonctions `create`, `is_empty`, `find` et `add` ? On pourra les exprimer en fonction de la hauteur des arbres donnés en entrée.

2 Arbres AVL

Étant donné que la complexité des opérations sur les arbres binaires de recherche dépend de la hauteur de l'arbre, il est intéressant de s'assurer que celle-ci soit la plus petite possible. On dit qu'un arbre binaire de recherche est un arbre AVL [1] si, pour tout nœud, la différence entre la hauteur du sous-arbre gauche et la hauteur du sous-arbre droit est -1, 0 ou 1. Comme on le montrera dans la question 9, cela permet de garantir que la hauteur de l'arbre est en $\Theta(\log n)$ où n est le nombre de nœuds. On appelle ***facteur d'équilibrage*** d'un nœud la différence entre la hauteur du sous-arbre gauche et la hauteur du sous-arbre droit. Ainsi, pour un arbre AVL, le facteur d'équilibrage de chaque nœud est -1, 0 ou 1. Par exemple, l'arbre de la figure 1 n'est pas un arbre AVL car le sous-arbre gauche du nœud `("ma_variable", 15)` est de hauteur 1 alors que son sous-arbre droit est de hauteur -1. Le facteur d'équilibrage de ce nœud est donc $2 \notin \{-1, 0, 1\}$. De même, l'arbre de la figure 2 n'est pas un arbre AVL car le facteur d'équilibrage du nœud `("ma_variable2", 16)` est -2. L'arbre de la figure 3 n'est pas un AVL alors que ceux des figures 4 et 5 sont des AVL.

2.1 Hauteur d'un arbre AVL

Le but de cette partie est de montrer que la hauteur d'un arbre AVL est de l'ordre de $\log(n)$ où n est le nombre de nœuds. Pour cela, on note $(f_n)_{n\in\mathbb{N}}$ la suite de Fibonacci définie par :

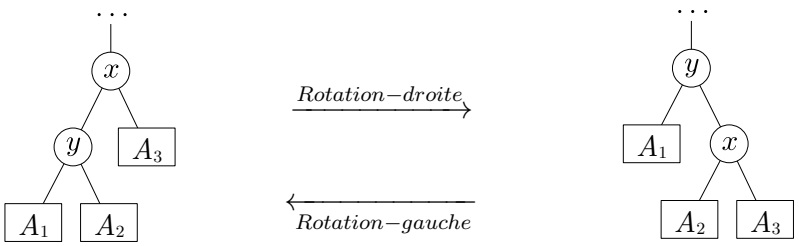
$$\begin{cases} f_0 = 1, f_1 = 1 \\ f_{n+2} = f_{n+1} + f_n \qquad \text{pour tout } n \geqslant 0 \end{cases}$$

7. Soit $h \in \{-1\} \cup \mathbb{N}$. Montrer qu'un arbre AVL de hauteur h possède au moins $f_{h+1} - 1$ nœuds.
8. Soit $\varphi = \dfrac{1+\sqrt{5}}{2} \approx 1.62$ le nombre d'or. On rappelle que φ est racine du polynôme $X^2 - X - 1$.
   (a) Montrer que $f_n \geqslant \varphi^{n-1}$ pour tout $n \in \mathbb{N}$.
   (b) On considère un arbre binaire de recherche de hauteur h avec n nœuds. Montrer que si cet arbre est un AVL, alors $h \leqslant \log_\varphi(n+1)$.
   (c) La réciproque de la proposition précédente est-elle vraie ?
9. En conclure que la hauteur d'un arbre AVL est en $\Theta(\log n)$.

1. Du nom de leur deux inventeurs : Georgii Adelson-Velsky et Evguenii Landis

2.2 Rotations dans un arbre AVL

Afin d'insérer ou de supprimer des nœuds dans un arbre tout en conservant la propriété AVL, on va avoir besoin de deux transformations appelées la *rotation-droite* et la *rotation-gauche*. Elles sont schématisées ci-dessous :



Arbre α Arbre β

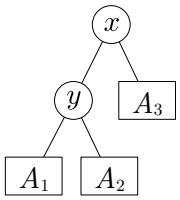
Ainsi, étant donné un nœud d'étiquette x dans un arbre, on peut appliquer une rotation-droite à ce nœud à condition que son sous-arbre gauche soit non vide. Si comme dans l'arbre α, on note y le fils gauche de x, A_1 le sous-arbre gauche de y, A_2 le sous-arbre droit de y et A_3 le sous-arbre droit de x ; alors une rotation-droite transforme l'arbre α en l'arbre β. Par exemple, appliquer une rotation-droite sur la racine de l'arbre figure 1 donne l'arbre de la figure 2 où :

x est l'étiquette `("ma_variable2", 16)`	y est l'étiquette `("ma_variable", 15)`
A_1 est le sous-arbre de racine `("a", -1)`	A_2 est l'arbre vide
A_3 est le sous-arbre de racine `("x0", 0)`	

De manière similaire, une rotation-gauche peut être appliquée à un nœud lorsque son sous-arbre droit est non vide. Par exemple, appliquer une rotation-gauche au nœud d'étiquette `("x0",0)` de l'arbre de la figure 1 donne l'arbre de la figure 3.

10. Montrer que si on applique une rotation-droite ou une rotation-gauche sur un nœud d'un arbre binaire de recherche, alors on obtient un autre arbre binaire de recherche.

On souhaite maintenant appliquer des rotations à un arbre binaire de recherche pour qu'il devienne un arbre AVL. Soit A un arbre ayant la forme ci-contre où x et y sont des étiquettes et A_1, A_2, A_3 sont des sous-arbres. On suppose que A est un arbre binaire de recherche, que A_1, A_2, A_3 sont des arbres AVL. Soit $h \geqslant 1$ l'entier tel que A_3 est de hauteur $(h-2)$.



Dans la suite, on se place dans l'un des deux cas suivants :

(cas 1) A_1 est de hauteur $(h-1)$ et A_2 est de hauteur $(h-2)$.

(cas 2) A_1 est de hauteur $(h-2)$ et A_2 est de hauteur $(h-1)$.

11. Montrer que :
    (a) Dans les deux cas, l'arbre n'est pas un arbre AVL.
    (b) Dans le cas 1, on peut obtenir un arbre AVL de hauteur h en appliquant une rotation.
    (c) Dans le cas 2, on peut obtenir un arbre AVL de hauteur h en appliquant deux rotations.

2.3 Insertion d'un nœud dans un arbre AVL

Dans cette partie est décrite la procédure pour insérer un nœud dans un arbre AVL. Une méthode similaire permet de supprimer un nœud, mais nous ne la détaillerons pas ici.

Description de la procédure d'insertion. Pour insérer un nœud z dans un arbre AVL, on procède en deux étapes :

- Le nœud z est inséré comme dans un arbre binaire de recherche standard. À la fin de cette étape, l'arbre peut ne plus être un arbre AVL.
- Des rotations sont appliquées sur l'arbre afin d'obtenir un arbre AVL. Pour choisir les rotations à effectuer, on remonte l'arbre à partir du père de z jusqu'à la racine. En d'autres termes on parcourt successivement tous les ancêtres de z. Étant donné x un ancêtre de z, il y a plusieurs cas :
  - Si le facteur d'équilibrage de x vaut -1, 0 ou 1, alors aucune rotation n'est effectuée à ce niveau.
  - Si le facteur d'équilibrage de x vaut 2 :
    - Dans le cas 1 de la partie 2.2, on applique la rotation donnée à la question 11b.
    - Dans le cas 2 de la partie 2.2, on applique les rotations données à la question 11c.
  - Si le facteur d'équilibrage de x vaut -2, on procède de manière symétrique par rapport au cas précédent.

Exemple. Prenons l'exemple de l'arbre AVL de la figure 4 dans lequel on souhaite ajouter un nœud d'étiquette `("b", 0)`. La première étape consiste à insérer le nœud comme dans un arbre binaire de recherche standard. On obtient alors l'arbre de la figure 1. On s'intéresse ensuite aux trois ancêtres du nouveau nœud :

- Le premier ancêtre est le nœud `("a", -1)`. Son facteur d'équilibrage est -1, il n'y a pas de rotation à effectuer.
- Le deuxième ancêtre est le nœud `("ma_variable", 15)`. Son facteur d'équilibrage est 2, il faut donc le rééquilibrer. On peut par exemple appliquer une rotation-gauche sur le nœud `("a", -1)`, puis une rotation-droite sur le nœud `("ma_variable", 15)` pour obtenir l'arbre de la figure 5.
- Le troisième ancêtre est le nœud `("ma_variable2", 16)`. Son facteur d'équilibrage est 0, il n'y a pas de rotation à effectuer.

Preuve de la correction de la procédure d'insertion. On souhaite montrer qu'après application de la procédure d'insertion, on obtient un arbre AVL. Montrons par récurrence sur $h \in \{-1\} \cup \mathbb{N}$ la propriété $\mathcal{P}_h$:

$$(\mathcal{P}_h) : \begin{cases} \text{Si on insère un nœud dans un arbre AVL de hauteur } h \text{ avec la procédure décrite} \\ \text{ci-dessus, alors l'arbre obtenu est un arbre AVL de hauteur } h \text{ ou } (h+1). \text{ Dans le} \\ \text{cas où la hauteur de l'arbre obtenu est } (h+1), \text{ aucune rotation n'a été effectuée.} \end{cases}$$

12. Montrer le cas de base $h = -1$.

Soit $h \geqslant 0$. Supposons $(\mathcal{P}_{h'})$ pour tout $h' < h$ et montrons $(\mathcal{P}_h)$. On considère A un arbre AVL de hauteur h. Soit G le sous-arbre gauche de A, soit D son sous-arbre droit et soient h_G et h_D leurs hauteurs. Sans perte de généralité, on suppose que la procédure d'insertion ajoute le nœud z dans le sous-arbre gauche (le cas où le nœud est ajouté dans le sous-arbre droit se traite de manière similaire). Une fois que le nœud a été inséré, on parcourt les ancêtres de z afin d'appliquer les rotations. Soit A' l'arbre obtenu juste avant de vérifier le facteur d'équilibrage de la racine. Notez que A et A' ont la même racine x et le même sous-arbre droit D. Soit G' le sous-arbre gauche de A', soit h' la hauteur de A' et soit $h_{G'}$ la hauteur de G'. D'après l'hypothèse de récurrence, G' est un arbre AVL de hauteur $h_{G'} \in \{h_G, h_G + 1\}$.

13. Dans le cas où $h_{G'} = h_G$, montrer que la procédure d'insertion n'applique pas de rotation au niveau de la racine et renvoie donc A' qui est un AVL de hauteur h.

14. Dans cette question, on suppose $h_{G'} = h_G + 1$. Puisque A est un AVL, on a

$$h_D \in \{h_G - 1, h_G, h_G + 1\}.$$

On traite ces trois cas séparément :

(a) Si $h_D = h_G$, montrer que la procédure d'insertion n'applique pas de rotation au niveau de la racine et renvoie donc A' qui est un AVL de hauteur $(h + 1)$.

(b) Si $h_D = h_G + 1$, montrer que la procédure d'insertion n'applique pas de rotation au niveau de la racine et renvoie donc A' qui est un AVL de hauteur h.

(c) Si $h_D = h_G - 1$, montrer que l'on se trouve dans le cas 1 ou le cas 2 de la partie 2.2 et que la procédure d'insertion renvoie un AVL de hauteur h.

On a donc montré $(\mathcal{P}_h)$ ce qui permet de conclure que la procédure d'insertion est correcte.

2.4 Implémentation.

15. Écrire une fonction « `est_AVL: dict -> bool` » qui teste si un arbre est un AVL. Ne pas oublier de vérifier que l'arbre est bien un arbre binaire de recherche. La fonction devra être de complexité linéaire en le nombre de noeuds dans l'arbre.

Afin de manipuler des arbres AVL en OCaml, il est pratique de retenir pour chaque nœud la hauteur du sous-arbre ayant ce nœud pour racine. On ajoute donc un nouveau champ nommé `hauteur` dans les étiquettes des nœuds :

```
type etiquetteAVL = {
    cle: string;
    valeur: int;
    hauteur: int;
  };;
```

```
type dictAVL =
  | Vide
  | N of etiquetteAVL * dictAVL * dictAVL;;
```

16. Écrire une fonction « `addAVL: dictAVL -> string -> int -> dictAVL` » qui ajoute un nœud dans un arbre AVL. Dans cette question, on suppose que dans l'arbre donné en entrée, les valeurs des champs `hauteur` sont correctes. De plus, vous devez faire en sorte que ces valeurs soient correctes dans l'arbre renvoyé par votre fonction.

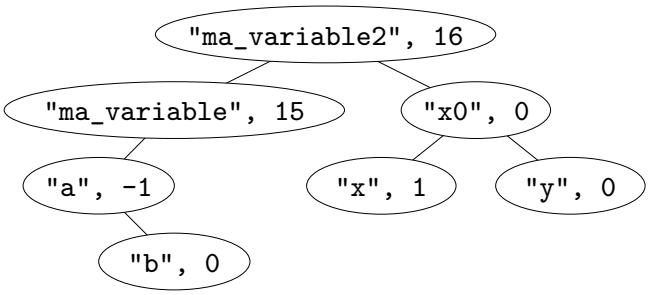


FIGURE 1

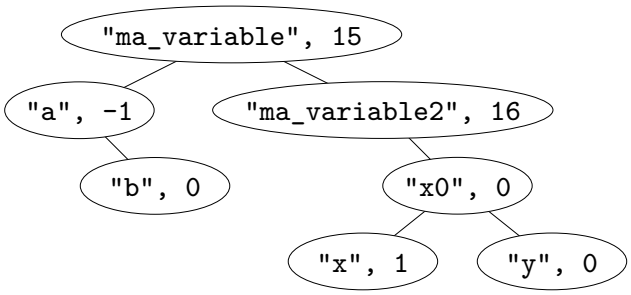


FIGURE 2

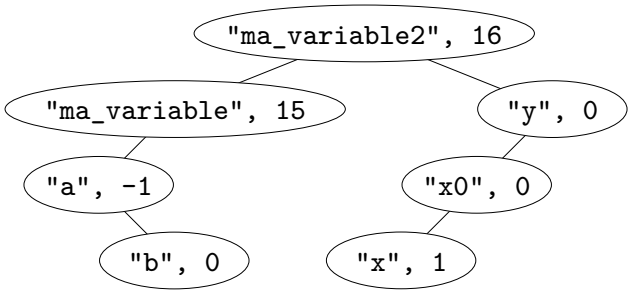


FIGURE 3

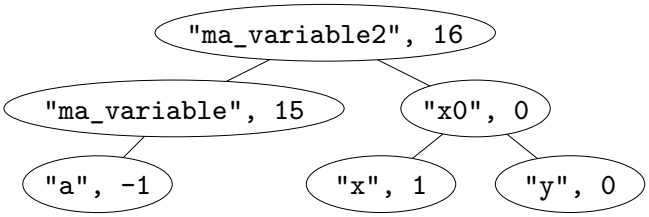


FIGURE 4

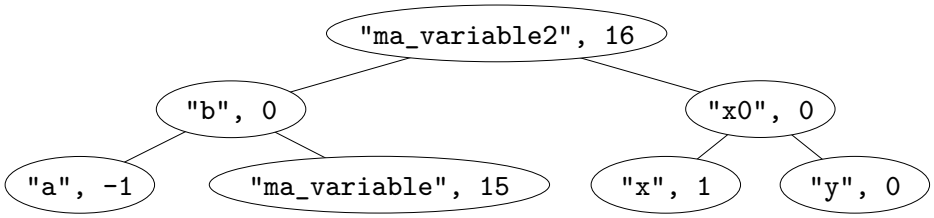


FIGURE 5