Projet d'Algorithmique et Programmation

AP3

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

Ce projet est à réaliser à deux ou trois. Il est à rendre au plus tard le 16 décembre 2022 avant 20h par le dépôt d'une archive au format `zip` à l'emplacement prévu sur UPdago.

Vous pouvez constituer votre groupe comme vous le souhaitez, mais **l'inscription dans un groupe sur UPdago est obligatoire** (voir dans l'onglet « Le projet » sur UPdago).

Nous avons en cours et en TD que les arbres binaires de recherche peuvent dégénérer en arbres filiformes et que, dans ce cas, on perd tous les avantages en termes de complexité en temps sur les opérations de base de recherche, d'insertion et de suppression.

Nous avons vu aussi en cours avec les arbres AVL qu'une façon de remédier à ce problème est de maintenir l'équilibre d'un arbre lors des opérations d'insertion et de suppression en effectuant des opérations de rotation.

Un autre moyen d'éviter les cas d'arbres de recherche dégénérés est d'augmenter les possibilités de direction de recherche et le possibilités d'insertion d'un élément. Pour cela, on introduit des arbres de recherche qui ne sont plus des arbres binaires, mais qui peuvent contenir aux noeuds un nombre variables de valeurs et de sous-arbres.

De tels arbres sont usuellement utilisés pour créer les index des tables dans les bases de données. Vous en verrez des exemples au second semestre dans le cours de Bases de Données.

1 Arbres $2-3-4$

Le projet vous propose d'étudier les arbres $2-3-4$. Un noeud d'un arbre $2-3-4$ peut contenir entre une et trois valeurs rangées en ordre croissant et ainsi avoir entre deux et quatre sous-arbres. De cette façon, lors de la recherche d'un élément après comparaison avec les valeurs en un noeud, il y a entre deux et quatre directions de recherche ce qui permet de trouver plus rapidement un élément. La figure 1 montre un exemple d'arbre $2-3-4$.

Le fait d'avoir plus de sous-arbres possibles permet aussi d'ajouter une contrainte, toutes les feuilles d'un arbre $2-3-4$ doivent être au même niveau. Ainsi la hauteur d'un arbre $2-3-4$ est toujours en fonction du logarithme de sa taille (i.e. le nombre de valeurs qu'il contient). Pour s'assurer de cette contrainte sur le niveau des feuilles, il faut effectuer des opérations de rotation qui peuvent se faire au cours des insertions ou des suppressions.

Plus précisément nous avons la définition suivante

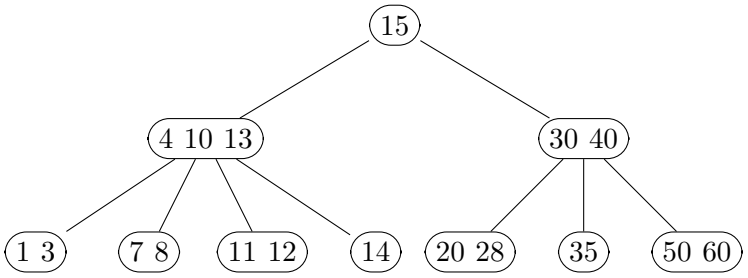


FIGURE 1 – Un exemple d'arbre $2-3-4$ de recherche.

Définition. *Un* **arbre $2-3-4$ de recherche** *est tel que chaque noeud contient un k-uplet avec $k = 1, 2, 3$, les valeurs $x_1, x_2, \ldots, x_k$ de ce k-uplet sont distinctes et ordonnées et chaque noeud possède $k+1$ sous-arbres $f_1, f_2, \ldots, f_{k+1}$ tels que*

- *Toutes les valeurs des noeuds de f_1 sont strictement inférieures à x_1*
- *Toutes les valeurs des noeuds de f_j avec $2 \leq j \leq k$ sont strictement supérieures à x_{j-1} et strictement inférieures à x_j.*
- *Toutes les valeurs des noeuds de f_{k+1} sont strictement supérieures à x_k.*

De plus, toutes les feuilles d'un arbre $2-3-4$ sont au même niveau et l'arbre vide est un arbre $2-3-4$. Selon le nombre de sous-arbres d'un noeud, on dira que celui-ci est un 2-noeud, un 3-noeud ou un 4-noeud.

Remarque. Dans les représentations graphiques, les arbres vides qui apparaissent aux feuilles ne sont pas indiqués.

▸ **Question 1.** *Dessinez trois exemples d'arbre $2-3-4$ de recherche.*

On a la propriété suivante sur la hauteur $h(t)$ d'un arbre $2-3-4$ t contenant n valeurs :

$$\log_4(n+1) \leq h(t) + 1 \leq \log_2(n+1)$$

▸ **Question 2.** *Démontrez cette propriété.*

▸ **Question 3.**

1. *En `OCaml` définissez un type somme `'a t_234tree` qui permet de représenter les arbres $2-3-4$.*
2. *Donnez la représentation de vos exemples de la question 1.*

▸ **Question 4.** *La recherche d'une valeur dans un arbre $2-3-4$ suit le même principe que pour un arbre binaire de recherche.*

1. *Écrivez la fonction `t234_search` qui recherche une valeur dans un arbre $2-3-4$.*

2. *Donnez un argument sur la complexité de cette fonction comptée en nombre de comparaisons.*

Dans un arbre $2-3-4$ de recherche, l'insertion se fait aux feuilles. Tant que l'on peut ajouter une valeur dans un noeud (*i.e.*ce n'est pas un 4-noeud) tout va bien. Lorsque l'on doit ajouter une valeur à un 4-noeud, il faut d'abord l'éclater, réorganiser localement l'arbre et ajouter la valeur. Les éclatements de 4-noeuds peuvent se propager un cascade jusqu'à la racine de l'arbre ce qui entraîne une perte d'efficacité de l'opération d'insertion.

Pour éviter ce phénomène de remontée en cascade des éclatements, on va systématiquement éclater les 4-noeuds rencontrés lors la recherche de l'emplacement d'ajout de la valeur dans l'opération d'insertion. Cela se fait en considérant les sous-arbres d'un 2-noeud ou d'un 3-noeud.

Par exemple, si le noeud considéré est un 2-noeud et que l'insertion doit se faire dans le sous-arbre gauche (*i.e.*la valeur à insérer est plus petite que la valeur du 2-noeud), si la racine du sous-arbre gauche est un 4-noeud, il faut l'éclater comme cela est illustré sur la figure 2.

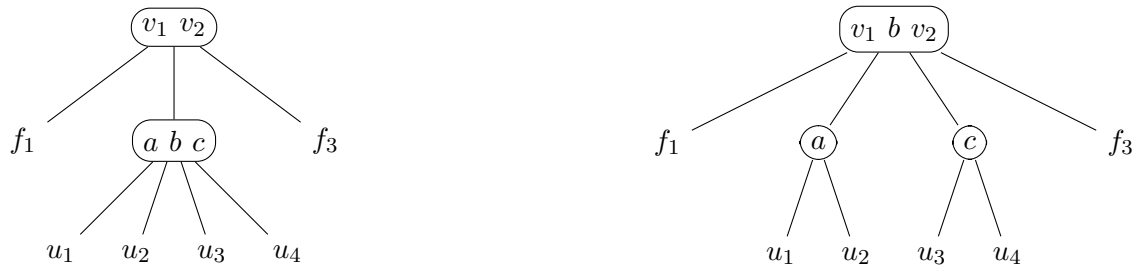
(a) Avant l'éclatement du 4-noeud.

(b) Après l'éclatement du 4-noeud.

FIGURE 2 – Éclatement d'un 4-noeud dont le noeud père est un 2-noeud.

On a un cas symétrique quand l'insertion doit se faire dans le sous-arbre droit du 2-noeud.

Pour un 3-noeud la situation est équivalente. Par exemple, si l'insertion doit se faire dans le sous-arbre au milieu du 3-noeud (*i.e.*celui qui numéroté f_3), on a l'éclatement tel qu'illustré sur la figure 3.



(a) Avant l'éclatement du 4-noeud.

(b) Après l'éclatement du 4-noeud.

FIGURE 3 – Éclatement d'un 4-noeud dont le noeud père est un 3-noeud.

▸ **Question 5.** *Dessinez les autres cas d'éclatement d'un 4-noeud.*

▸ **Question 6.** *Écrivez une fonction* `t234_insert : 'a t_234tree * 'a -> 'a t_234tree` *qui insère une valeur dans un arbre* $2-3-4$ *(on suppose cette valeur différentes de toutes les valeurs déjà contenue dans l'arbre).*

▸ **Question 7.** *Testez votre fonction d'insertion en insérant successivement les valeurs* $4, 35, 10, 13, 3, 30, 15, 12, 7, 40, 20, 11, 6$. *Vérifiez (« à la main ») à chaque insertion que vous avez bien un arbre* $2-3-4$ *de recherche.*

2 Arbres Rouge-Noir

Lorsque l'on veut implanter les arbres $2-3-4$ de recherche dans un langage de programmation où l'on n'a pas la possibilité d'utiliser des types sommes (ou bien lorsque ceux-ci sont peu utilisés comme, par exemple, en C), il est possible de représenter ces arbres par des arbres rouge-noir qui sont des arbres binaires que l'ons ait facilement implanter avec des pointeurs.

Dans cette partie, nous allons implanter ces arbres rouge-noir avec un type somme afin de limiter la complexité du projet. Nous allons aussi voir comment transformer un arbre $2-3-4$ de recherche en un arbre rouge-noir et réciproquement.

Définition. *Un arbre rouge-noir est un arbre binaire auquel on ajoute aux noeuds soit la couleur, soit la couleur noire. De plus,*

1. *pour un noeud toutes les valeurs de son sous-arbre gauche sont strictement inférieures à la valeur du noeud et toutes les valeurs de son sous-arbre gauche sont strictement supérieures à la valeur du noeud.*
2. *Le parent d'un noeud rouge ne peut pas être un noeud rouge.*
3. *Tout chemin de la racine à une feuille contient le même nombre de noeuds noirs.*

De plus l'arbre vide est considéré comme un noeud de couleur noire.

Remarque. Par définition un arbre rouge-noir est un arbre binaire de recherche.

▸ **Question 1.** *En prenant trois exemples d'arbres rouge-noir, convainquez-vous qu'un tel arbre est équilibré. Est-il H-équilibré ? Si oui donnez un argument pouvant permettre d'établir ce fait, si non donnez un contre-exemple.*

▸ **Question 2.**

1. *En* `OCaml` *définissez un type somme* `'a t_rbtree` *qui permet de représenter les arbres rouge-noir.*
2. *Donnez la représentation de vos exemples de la question 1.*

L'insertion d'une valeur dans un arbre rouge-noir suit les mêmes principes que l'insertion d'une valeur dans un arbre binaire de recherche. Cependant, pour conserver un arbre rouge-noir, il faut

1. imposer qu'après l'insertion la racine soit de couleur noire.
2. imposer que l'insertion d'une valeur dans un arbre vide génère un noeud de couleur rouge.
3. que l'arbre soit rééquilibré lors de l'insertion dans un sous-arbre.

En colorant les nouveaux noeuds en rouge (*i.e.*ceux qui sont générés en insérant une valeur dans l'arbre vide), on garantit que la troisième condition de la définition est vérifiée. Cependant, il se peut que la deuxième ne le soit plus. Une ananlyse des cas qui peuvent se produire montre qu'il y a quatre cas problématiques. Ces quatre cas sont illustrés sont la figure 4 où les noeuds dessinés en carré sont noirs et ceux dessinés en cercle sont rouges.

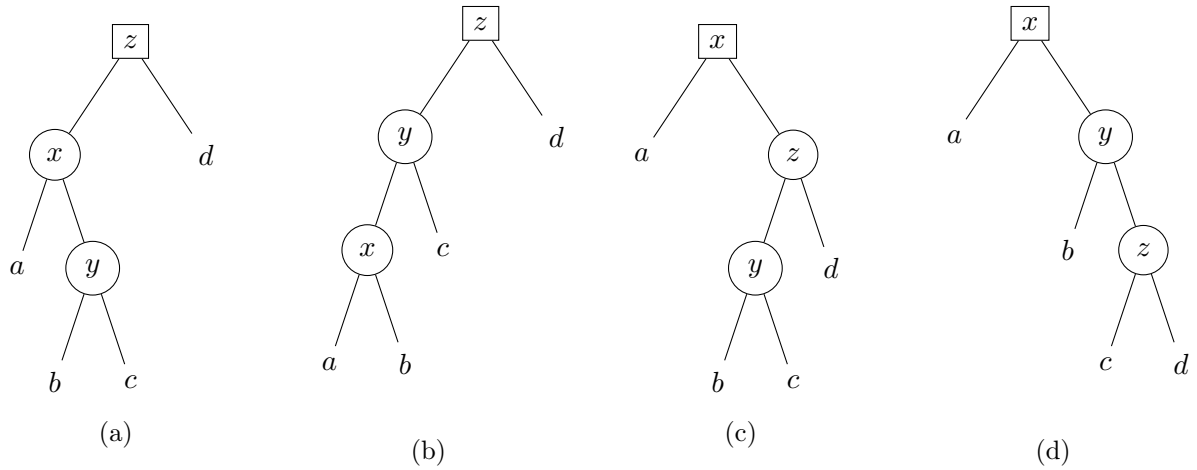


FIGURE 4 – Les quatre cas problématiques lors de l'insertion dans un arbre rouge-noir.

Ces quatre cas s'éliminent de la même façon en effectuant la rotation indiquée sur la figure 5 qui permet à la fois de rééquilibrer localement l'arbre et de rétablir la propriété d'être un arbre rouge-noir.

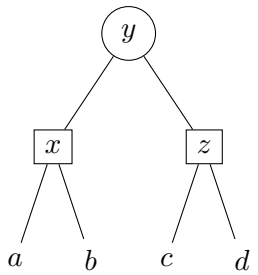


FIGURE 5 – La rotation qui permet de rééquilibrer localement l'arbre et de rétablir la propriété d'être un arbre rouge-noir.

▸ **Question 3.** *Écrivez une fonction* `rb_balance : color * 'a * 'a t_rbtree * 'a t_rbtree -> 'a t_rbtree` *qui reconstruit un arbre rouge-noir à partir d'une couleur, d'une valeur, d'un sous-arbre gauche et d'un sous-arbre droit. Il vous faudra analyser (i.e. filtrer) les différents cas qui peuvent se produire pour le sous-arbre gauche et le sous-arbre droit.*

Lors des opérations de rééquilibrage pendant l'insertion d'une valeur, il se peut que la racine devienne un noeud rouge qui sera le parent d'un noeud rouge. Pour éviter cela, il suffit d'imposer que la racine soit un noeud noir à la fin de la s'insertion.

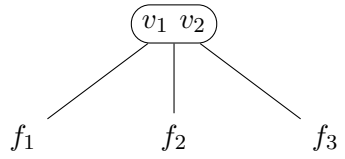
▸ **Question 4.** *Écrivez une fonction* `rb_insert : 'a t_rbtree * 'a -> 'a t_rbtree` *qui insère une valeur dans un arbre rouge-noir (on suppose cette valeur différentes de toutes les valeurs déjà contenue dans l'arbre).*

▸ **Question 5.** *Testez votre fonction d'insertion en insérant successivement les valeurs* $4, 35, 10, 13, 3, 30, 15, 12, 7, 40, 20, 11, 6$. *Vérifiez (« à la main ») à chaque insertion que vous avez bien un arbre rouge-noir.*

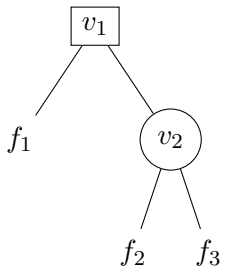
Pour pouvoir implanter les arbres $2-3-4$ avec les arbres rouge-noir, il faut traduire un arbre $2-3-4$ en un arbre rouge-noir et réciproquement. on dit que l'on a un isomorphisme entre ces deux structures de données.

La traduction se fait on considérant que les k-uplets contenus dans les k noeuds forment des arbres et en coloriant leurs noeuds de façon à conserver la structure d'un arbre $2-3-4$.

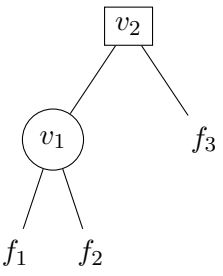
La traduction d'un 2-noeud d'un arbre $2-3-4$ se fait directement en coloriant ce noeud en noir. Pour la traduction d'un 3-noeud, il faut considérer deux cas, soit la première valeur du noeud est plus grande que la seconde, soit c'est le contraire. Chacun de ces cas est illustré sur la figure 6.



(a) Le 3-noeud à transformer



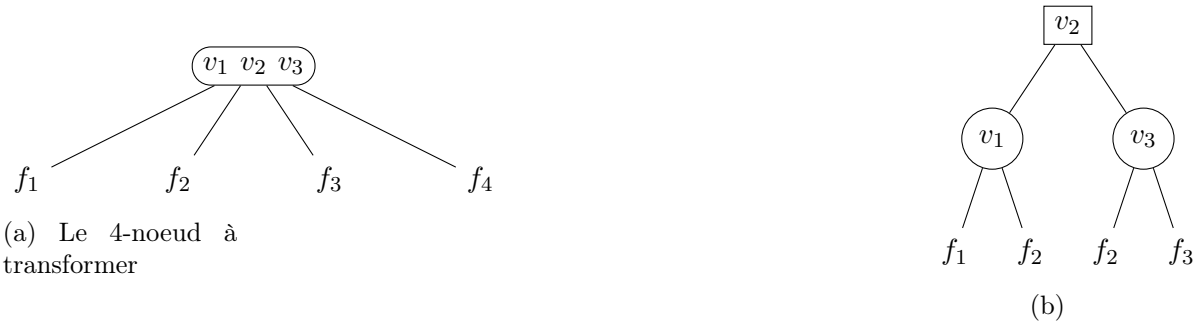
(b) Cas où $v_1 > v2$.



(c) Cas où $v_1 < v_2$

FIGURE 6 – Transformation d'un 3-noeud en un arbre rouge-noir.

La traduction d'un 4 noeud se fait directement en coloriant en noir le noeud issu de la deuxième valeur du noeud et en rouge les noeuds issus des deux autres valeurs. Comme cela est illustré sur la figure 7



(a) Le 4-noeud à transformer

(b)

FIGURE 7 – Transformation d'un 4-noeud en un arbre rouge-noir.

▸ **Question 6.** *Écrivez une fonction* `to_rb : 'a t_234tree -> 'a t_rbtree` *qui transforme un arbre* $2-3-4$ *de recherche en un arbre rouge-noir.*

▸ **Question 7.** *Écrivez une fonction* `to_234 : 'a t_rbtree -> 'a t_234tree` *qui transforme un arbre rouge-noir en un arbre* $2-3-4$ *de recherche. Il vous faudra faire attention aux différents cas où les sous-arbres sont vide pour construire correctement les différents* k*-noeuds.*

▸ **Question 8.** *(Question facultative) Si vous observez les arbres obtenus lors des insertions en suivant la liste donnée en exemple, vous verrez que les traductions des arbres* $2-3-4$ *en arbres rouge-noir ne sont pas identiques aux arbres rouge-noir que l'on obtient en faisant les mêmes insertions. Essayez d'expliquer pourquoi.*

3 Les bonus

Si vous répondez entièrement et correctement à toutes les questions précédentes, vous avez un projet qui obtiendra la mention « Bien ». Pour obtenir la mention « Très Bien » vous devez répondre à l'une des deux questions suivantes.

▸ **Question 1.** *Pour chacune des structures de données, les arbres* $2-3-4$ *et les arbres rouge-noir, détaillez les algorithmes de suppression d'une valeur en un noeud et codez les.*

▸ **Question 2.** *Effectuez une expérimentation convaincante des problèmes de déséquilibre lors de l'insertion de (nombreuses) valeurs aléatoires dans des arbres binaires de recherche. Votre expérimentations doit porter sur plusieurs milliers d'arbres binaires de recherche comprenant plusieurs milliers (voir plus) de valeurs différentes. Vous répéterez suffisamment de fois les mêmes expériences afin d'obtenir des moyennes significatives. Vous répéterez les mêmes expérimentations sur les arbres rouges-noirs afin de constater expérimentalement que la recherche d'une valeur se fait une fonction logarithmique de la taille de l'arbre.*