

Arbres Rouge-Noirs

L'objectif de ce TP est d'obtenir une implémentation complète et persistante¹ des arbres rouge-noir en OCaml : test d'appartenance, insertion. L'ajout de la suppression donne la version étoilée du TP.

A Introduction

A.1 Fonctionnement du TP

A.1.i : Todos

L'implémentation intégrale des ARN est un peu longue. Aussi avec ce TP vous est fourni un squelette : `arn.ml`. Il contient certaines fonctions pré-implémentées et d'autres contenant des `failwith "TODO"`. Votre travail est de remplacer ces `failwith "TODO"` par du code correct (et éventuellement de changer le filtrage par motif associé).

Attention, vous croiserez aussi d'autres `failwith` : ceux-ci ne sont pas des trous à compléter, ce sont des messages d'erreur (souvent associés à des cas impossibles) qui sont là pour aider au débogage.

Par exemple, la fonction `etiquette` contient `| Nil -> failwith "etiquette Nil"`. Ce n'est pas un trou à compléter, c'est une erreur pour vous aider à déboguer : vous avez voulu accéder à l'étiquette de Nil qui n'en a pas.

A.1.ii : Interface

En plus du squelette, vous trouverez aussi `arn_doc.mli`. C'est un fichier interface (comme les headers en C) qui contient la documentation des fonctions.

Pour en créer un format plus agréable à lire, je recommande l'utilisation de `ocamldoc` : cette commande terminal transforme un `mli` en page html. Pour l'utiliser, ouvrez un terminal et allez dans le dossier du `mli` puis lancez : `ocamldoc arn_doc.mli -html`. Vous obtiendrez beaucoup de fichiers html, dont `Arn_doc.html` : c'est ce dernier qui vous intéresse ! Vous n'avez plus qu'à l'ouvrir avec votre navigateur, par exemple `firefox Arn_doc.html`.

Si vous voulez que ces html soient générés ailleurs, regardez l'option `-d` de `ocamldoc`.

Enfin, vous trouverez `ex.ml` : c'est un fichier qui contient des exemples bien choisis d'ARN. Vous devriez l'utiliser pour tester vos fonctions. Vous devriez aussi ne pas vous limiter à ces exemples, et en faire d'autres qui correspondent à vos besoins.

A.1.iii : Let's go

Votre mission, et vous l'acceptez, est de compléter les trous de `arn.ml` afin de respecter la spécification indiquée dans `arn_doc.mli`. En particulier, à la fin les fonctions de Recherche, d'Insertion (et éventuellement de Suppression) doivent fonctionner².

1. Aussi appelé « structure de données fonctionnelle », c'est à dire que les insertions/suppressions ne modifient pas en place un ARN, mais en renvoient un nouveau.

2. Attention, il ne s'agit pas simplement de réussir à Rechercher et Insérer, mais aussi et surtout de le faire dans des Rouge-Noirs.

A.2 Définitions

On utilisera la définition suivante d'Arbres Rouges Noirs :

Définition 1.

Un ARN arbre binaire de recherche, aux étiquettes deux à deux distinctes (pas de doublon). On ajoute en outre une couleur à chaque noeud : Rouge ou Noir. Il doit vérifier les conditions suivantes :

- (0) C'est un Arbre Binaire de Recherche, c'est à dire que le parcourt infixe lit les étiquettes par ordre strictement croissant.
- (1) Aucun noeud Rouge n'a de fils Rouge.
- (2) Tous les chemins de la racine à un Nil contiennent le même nombre de noeuds noirs (sans compter la racine).
- (3) La racine est Noire.
- (4) Les Nils sont Noirs.

Remarque : la propriété (3) empêche de définir les ARN par induction. Pour désambigüiser, dans ce sujet on appelle :

- *arbre rouge-noir correct* un arbre vérifiant toutes les conditions.
- *sous-arbre rouge-noir correct* un arbre vérifiant toutes les conditions sauf éventuellement la (3).
- *sous-arbre rouge-noir presque correct en Rouge* un arbre vérifiant les conditions (0), (2), (et éventuellement (3)). En outre, s'il ne vérifie pas (1), alors cette dernière est rompue une seule fois (il existe un unique enchainement de noeuds Rouges) **et** cette rupture à la racine (la racine est Rouge, et a un enfant qui est rouge).

Dans ce sujet, on utilise le type OCaml suivant pour représenter les couleurs. Vous pouvez pour l'instant ignorer la couleur Doublenoir :

```
1 type couleur = Rouge
2               | Noir
3               | Doublenoir (* état temporaire de la suppression *)
```



À l'aide de ce type `Couleur`, on définit de la manière suivante un arbre Rouge-Noir. Notez que d'après (3), la couleur des Nil devra toujours être Noir. (Sauf éventuellement durant la suppression.)

```
1 type 'a arn = Nil of couleur (* Noires sauf temporairement
2                               durant la suppression *)
3               | Node of couleur * 'a arn * 'a * 'a arn
```



Vous noterez que c'est un type immuable : nos fonctions ne devront pas *modifier* un arbre, mais bien en renvoyer un nouveau sur lequel a été appliquée l'opération voulue. Afin de simplifier ses phrases, cet énoncé mélangera parfois ces deux notions.

Pour compiler et tester vos fonctions, vous avez deux options :

- Travailler en mode compilé. Dans ce cas le fichier `ex.ml` devrait débiter par `open Arn`. Pour rappel, cela donne accès aux fonctions de `arn.ml` sans avoir à ajouter le préfixe `Arn.` : ainsi, si `arn.ml` définit `etiquette`, alors `ex.ml` pourra utiliser cette fonction en l'appelant simplement `etiquette` (et non `Arn.etiquette`).

Pour compiler vos deux fichiers ensembles : `ocamlopt -o exec arn.ml ex.ml`

- Travailler en mode interprété. En ouvrant `utop` depuis le dossier de vos fichiers, `#use "file.ml"` permet de compiler `file.ml` et de charger ses fonctions dans l'interpréteur.

L'interpréteur peut-être plus pratique dans un premier temps : l'affichage des arbres est fort commode.

Un dernier conseil pour la route :

FAITES. DES. SCHÉMAS!!

B Mise en jambe

1. Représentez l'arbre `arn_ex` de `ex.ml` sur un schéma.

On s'intéresse à la partie « Accesseurs ». Elle a deux sous-parties³ : « Propriétés d'un noeud » et « Propriétés d'un arbre ».

Cette partie vise à vous familiariser avec le type.

2.
 - a. Complétez les codes de la sous-partie « Propriétés d'un noeud ».
 - b. Complétez les codes de la sous-partie « Propriétés d'un arbre ». C'est notamment ici que se trouvent les fonctions de recherche. Pour `hauteur_noire`, on utilise la définition du cours : nombre d'arcs *entrants* dans un noeud Noir sur un chemin de la racine à Nil. Considérez qu'un noeud Noir porte un niveau de Noir et qu'un noeud Doublenoir en porte 2 (et un Nil doublenoir en porte 1). Vous prendrez en compte la couleur des Nil.
 - c. Testez vos fonctions.

C Insertion

Le principe de l'insertion est celui vu dans le cours : on crée une nouvelle feuille rouge avec la clé à insérer, puis on corrige les problèmes en remontant jusqu'à la racine.

On utilise ici une méthode légèrement meilleure que celle vue en cours, car elle permet parfois de réparer localement l'enchaînement de Rouge sans remonter jusqu'à la racine. Les schémas de cet énoncé sont issus de <https://www.irif.fr/~carton/Enseignement/Algorithmique/Programmation/RedBlackTree/>.

La situation à réparer est toutefois la même : un noeud rouge p qui a un enfant rouge x . On nomme pp le parent de p , et f l'adelphe de p .

En d'autres termes :

- f est racine d'un sous-arbre rouge-noir correct.
- p est la racine d'un sous-arbre rouge-noir presque correct en rouge
- pp est Noir, et ses deux sous-arbres ont même hauteur noire.

Dans toute cette partie, la question de la couleur des Nil ne se pose pas : vous pouvez matcher tous les Nil sur `Nil _`, et lorsque vous créez un Nil respectez (3) en ne créant que `Nil Noir`. Ainsi, tous vos Nil seront toujours Noirs.

C.1 Recoloriages et Rotations

Afin de corriger les problèmes, il faut pouvoir appliquer des recoloriage et des rotations.

3. Complétez les trous des fonctions `devient_rouge` et `devient_noir`. (Pour les Nil, souvenez-vous qu'il est interdit de colorier un Nil en Rouge. Ignorez les fonctions `noircir` et `eclaircir`.)

On rappelle le principe des rotations :

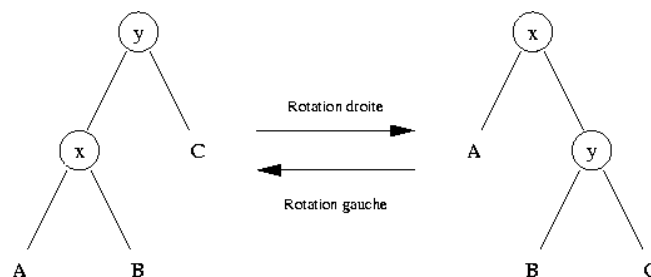


FIGURE XVII.1 – Rotation droite et gauche

4. Compléter les trous de la fonction `rotation_droite`.

³ Cette notion de parties/sous-parties se voit très bien dans le html produit par `ocamlDoc`.

C.2 Cas d'insertion

On distingue plusieurs cas (Les noeuds sans couleur sur le schéma sont Noirs, et les rouges sont... Rouges) :

N°	Conditions	Schéma	Solution
1	f est Rouge .	<p>(ou symétrique avec x enfant droit et/ou p enfant droit.)</p>	Colorier p et f en Noir, et pp en Rouge.

TABLE XVII.1 – Insertion : cas 1

N°	Conditions	Schéma	Solution
2	Dans tous les cas 2, p est fils gauche et f est Noir.		
2.a	x est le fils gauche de p .		Faire une rotation droite à la racine (pour faire remonter p en nouvelle racine et envoyer pp à sa droite). Ensuite, recolorier p en Noir et pp en Rouge.
2.b	x est le fils droit de p .		On se ramène au cas 2.a . Pour cela, on effectue une rotation gauche sur le sous-arbre enraciné en p (pour faire remonter x et envoyer p à sa gauche).

TABLE XVII.2 – Insertion : cas 2

5. Complétez les trous de la fonction `corrige_rouge` .
6. Testez. Testez. Testez. Testez.

N°	Conditions	Schéma	Solution
3	Dans tous les cas 3, p est fils droit et f est Noir.	Les cas 3 sont les symétriques des cas 2. Par symétrie, on entend la symétrie axiale par rapport à la droite verticale passant au milieu de pp. (D'où d'ailleurs la définition des cas 3 de la case précédente.)	
3.a	Si x est fils droit.	Faire le schéma!	Solution symétrique de 2.a.
3.b	Si x est fils gauche.	Faire le schéma!	Solution symétrique de 2.b. On va donc se ramener à 3.a.

TABLE XVII.3 – Insertion : cas 3 (symétrique de 2)

7. Complétez les schémas et les solutions des cas 3.

Maintenant, il s'agit d'écrire une fonction « aiguillage » : son travail est de prendre en argument un arbre dont un sous-arbre est presque correct en Rouge, déterminer dans lequel des cas précédents on se trouve, et d'appliquer la fonction `ins_casXX` associée afin de se ramener à un sous-arbre rouge-noir correct.

8. Complétez les trous de la fonction `corrige_rouge`.

C.3 Insertion en elle-même

Pour insérer, on va procéder de la manière suivante : on descend récursivement jusqu'à trouver où placer la nouvelle feuille Rouge contenant l'élément à insérer. Ensuite, on remonte récursivement en appliquant à chaque étape `corrige_rouge` : cela permet de faire remonter l'éventuel enchaînement de Rouge jusqu'à la racine initiale !

La fonction `insere_aux` effectue cela.

9. a. Complétez le cas de base de `insere_aux`.
- b. Lisez le reste du code. Comprenez-vous pourquoi il réalise bien la descente / remontée expliquée ci-dessus ? Pour vous en assurer, expliquez-la à votre mascotte comme si elle avait 5 ans.

Il reste un cas que nous n'avons pas traité : si l'enchaînement R-R a lieu à la racine.

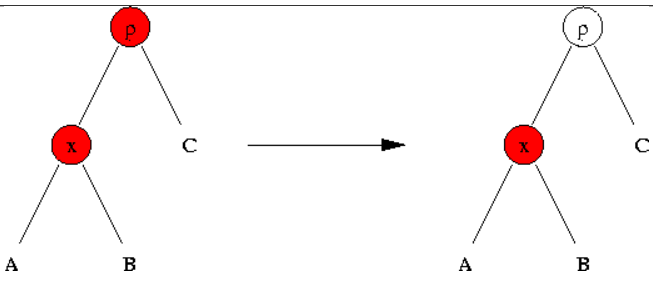
N°	Conditions	Schéma	Solution
0	p n'a pas de père : c'est la racine.		Colorier p en Noir, c'est à dire assurer (4).

TABLE XVII.4 – Insertion : cas 0

10. Complétez la fonction `insere`. Vous devrez vous assurer d'obtenir un arbre rouge-noir correct.
11. Testez. Testez. Testez. Testez.
12. a. Écrire une fonction `arn_of_list` : 'a list -> 'a arn qui prend en argument un tableau sans doublons et renvoie un arn contenant tous ses éléments.
- b. Écrire une fonction `sort_by_arn` qui trie un tableau en utilisant le parcours de l'arn associé au tableau.
- c. Bonus : ajoutez ces fonctions et leur documentation au `.mli`



FIGURE XVII.2 – Un pingouin très fier de vous.

D Suppression

Pour les plus motivé-es d'entre vous, nous allons passer à la suppression. C'est une opération plus délicate... que nous rendrons surprenamment accessible grâce à la couleur `Doublenoir`.

On utilise l'idée suivante : on supprime toujours un noeud qui n'a qu'un ou zéro enfants. Si le noeud cherché en a 2, on le remplace par le minimum de son sous-arbre droit et on va supprimer ce dernier (qui n'a pas d'enfant gauche).

L'enfant du noeud supprimé va alors « remonter » à sa place. En remontant, il va additionner la couleur du noeud supprimé à la sienne selon ces règles :

\	Rouge	Noir	Doublenoir
Rouge	Rouge	Noir	erreur
Noir	Noir	Doublenoir	error

TABLE XVII.5 – Règles d'addition de couleurs

Dit en termes mathématiques : l'addition est commutative, Rouge est neutre, Noir+Noir = Doublenoir, et tenter d'additionner Doublenoir fait une erreur.

Attention, si le noeud supprimé n'avait pas d'enfant, c'est une feuille qui remonte... on doit donc donner des couleurs aux feuilles. On ajoute donc la règle suivant à la def d'ARN : (4) *les Nil sont Noirs (ou temporairement DoubleNoir durant une insertion).*⁴

Avec cette façon d'insérer, le seul cas de rupture des propriétés d'ARN si et seulement si un doublenoir a été créé : le but sera de s'en débarrasser, soit en réglant localement le problème, soit en le faisant remonter jusqu'à la racine (que l'on peut recolorier en noir). On nommera :

- x le noeud doublenoir
- p son parent. On se placera toujours dans le cas où p est la racine du sous-arbre en cours.
- f l'adelphe de x . C'est donc l'autre enfant de p .
- g et d les éventuels enfants gauches et droits de f .

Vous ferez bien attention à ne pas mélanger avec les notations précédentes.

13. Faire un schéma de ces nouvelles notations.

On appellera :

- sous-arbre rouge-noir presque correct en Noir un arbre vérifiant les conditions (0), (2), (4) mais dont la racine peut-être Doublenoir.

Nous allons coder comme précédemment : auxiliaires simples, puis différents cas et aiguillage, puis enfin suppression.

D.1 Recoloriages Rotations

Les fonctions `noircir` et `eclair` sont supposées respectivement ajouter un Noir et soustraire un Noir.

14. a. Complétez ces fonctions.
 b. Complétez également `est_doublenoir`
 c. Si vous en ressentez le besoin, vous pouvez créer d'autres petites fonctions annexes. Par exemple, `devient_rouge` et `devient_noir`.

D.2 Cas de suppression

L'objectif est maintenant de coder `corrige_doublenoir`.

On distingue plusieurs cas. Attention, le code couleur des schémas est comme suit :

- Les noeuds pleins sont Rouges⁵
- Les noeuds vides sont Noirs.
- Les noeuds vides avec une bordure épaisse sont DoubleNoir.

N°	Conditions	Schéma	Solution
fNil	f est Nil.	Faire un schéma. Indiquez les hauteurs noires.	?

TABLE XVII.6 – Suppression : cas fNil

15. Complétez le tableau du cas Nil. En déduire `suppr_casfNil`.

4. Pourquoi Noirs et pas Rouge ? Parce que cela évite les enchainements R-R, et que si tous les Nils sont Noirs la condition sur la hauteur noire reste vraie.

5. *smiley_yeux_aux_ciel*

N°	Conditions	Schéma	Solution
1	Dans tous les cas 1, x est fils gauche et f est Noir.		
1.a	g et d sont Noirs.		Noircir p . Éclairir x et f .
1.b	d est Rouge.		On effectue une rotation gauche sur l'arbre (pour faire remonter f et envoyer p à sa gauche). Le noeud f prend la couleur du noeud p (qu'il remplace maintenant). Éclairir x (qui devient donc Noir). Colorier p et d en Noir.
1.c	g est Rouge et d est Noir.		On se ramène au cas précédent. On effectue une rotation droite sur le sous-arbre enraciné en f (pour faire remonter g et envoyer f à sa droite). Le noeud g devient Noir (l'ancienne couleur de f qu'il remplace) et f devient Rouge.

TABLE XVII.7 – Suppression : cas 1

16. À l'aide d'un papier et d'un crayon, convainquez-vous que les solutions proposées aux cas 1 permettent bien d'obtenir un sous-arbre rouge-noir presque correct en Noir *de même hauteur noire que l'original*.
17. Complétez les trous du code pour les fonctions `suppr_cas1XX`. ces fonctions prennent en entrée un arbre dont on garanti qu'il correspond au cas indiqué, et renvoie la correction donnée par l'énoncé.
18. Testez. Testez. Testez. Testez.

Le cas 2 est un peu particulier car il se réduit à un cas 1 - nous le traiterons plus tard. Passons au cas 3 : c'est le symétrique du cas 1 !

N°	Conditions	Schéma	Solution
3	Dans tous les cas 3 , x est fils droit et f est Noir.	<i>Les cas 3 sont les symétriques des cas 1. Par symétrie, on entend la symétrie axiale par rapport à la droite verticale passant au milieu de pp.</i>	
3.a	g et d sont Noirs.	Faire le schéma!	Solution symétrique de 1.a.
3.b	g est Rouge.	Faire le schéma!	Symétrique de 1.b.
3.c	g est Noir et d est Rouge.	Faire le schéma!	Symétrique de 1.c.

TABLE XVII.8 – Suppression : cas 3

19. Complétez le tableau des cas 3.

Passons enfin au cas 2 (et son symétrique le cas 4) :

N°	Conditions	Schéma	Solution
2	x est fils gauche et f est Rouge.		On va se ramener à un cas 1. On effectue une rotation gauche sur l'arbre (pour envoyer p à gauche et faire remonter f). On recolorie ensuite p en Rouge et f en Noir (x n'est pas modifié et reste Doublenoir). Le sous-arbre enraciné en p est maintenant dans le cas 1.
4	x est fils droit et f est Rouge.	Faire le schéma!	Symétrique de 2 : on va se ramener à un cas 3.

TABLE XVII.9 – Suppression : cas 2 et 4

On remarque donc que la correction de Doublenoir peut appeler les cas 2 ou 4, et que 2 et 4 peuvent appeler la correction : ces fonctions sont donc co-récurrentes !

20. Complétez la fonction `suppr_cas2` en supposant que `corrige_doublenoir` est totalement correcte.
21. `corrige_doublenoir` est une fonction « aiguillage » : elle détermine dans lequel des cas précédents on se trouve et applique la bonne fonction auxiliaire. Complétez ses trous.

D.3 Suppression en elle-même

Pour supprimer x , on va procéder ainsi : on descend dans l'arbre jusqu'à trouver x . Si x n'a qu'un ou zéro enfant, on fait remonter son enfant (ou un Nil) à sa place, et on additionne leurs couleurs. Sinon, on recherche l'étiquette minimale à droite de x notée m , on remplace l'étiquette x par m , puis on descend supprimer ce minimum. Quand on remonte, à chaque étape on corrige l'éventuel problème de Doublenoir.

La fonction `supprime_aux` effectue cela :

22. Complétez les cas de base de `supprime_aux` (ce sont les cas avec zéro ou un enfant non-Nil).
23. Lisez le reste du code. Comprenez-vous pourquoi il réalise bien la descente / remontée expliquée ci-dessus ? Expliquez à votre peluche comme si elle avait 5 ans.

Il un cas que nous n'avons pas traité :

N°	Conditions	Schéma	Solution
0	x n'a pas de père : c'est la racine.	$A \leftarrow A$	On éclaircit x .

TABLE XVII.10 – Suppression : cas 0

24. Complétez la fonction `supprime`. Vous devrez vous assurer d'obtenir un arbre rouge-noir correct.
25. Testez. Testez. Testez. Testez.



FIGURE XVII.3 – Un pingouin toujours très fier de vous.