TD n°18 – Corrigé

1 ABR

On va implémenter les opérations des ABR en Ocaml. On définit les arbres binaires de la manière habituelle, `type 'a arbre = Vide | N of 'a arbre * 'a arbre`.

1. Écrire une fonction `recherche:'a arbre-> 'a->bool` qui renvoie **`true`** si un ABR contient une certaine étiquette e et **`false`** sinon. On devra garantir une complexité en $O(h(a))$ pour cette fonction.
   On applique la métode vue en cours.

```
let rec recherche a e = match a with
  |Vide -> false
  |N(g,x,d) -> if x=e then true
               else if x<e then recherche g e
               else recherche d e;;
```

2. Écrire une fonction `supprime:'a arbre-> 'a->'a arbre` qui supprime une étiquette e d'un ABR (on suppose que e apparait au plus une fois). Le résultat doit être un ABR et la fonction doit avoir une complexité en $O(h(a))$.
   On applique la métode vue en cours.

```
let supprime a e =
  let rec plus_a_gauche a = (*supprime le noeud le plus a gauche, renvoie le nouvel
  arbre et l'étiquette supprimée*)
    match a with
      |Vide -> failwith "pas normal"
      |N(Vide,x,d) -> d, x
      |N(g,x,d) -> let aa,et = plus_a_gauche g in N(aa,x,d),et
  in
  let rec aux a = match a with
      |Vide -> failwith "pas trouvé"
      |N(g,x,d) when x<e -> N(aux g,x,d) (*recherche du noeud e*)
      |N(g,x,d) when x>e -> N(g,x,aux d) (*recherche du noeud e*)
      |N(g,x,d) when x=e -> match g,d with
                           |Vide, Vide -> Vide (*0 enfants*)
                           |Vide, d -> d (*1 enfant*)
                           |g, Vide -> g (*1 enfant*)
                           |g,d -> let dd, et = plus_a_gauche d in (*2 enfants*)
                                   N(g,et,dd)
  in aux a;;
```

3. Écrire une fonction `est_abr:'a arbre->bool` qui vérifie si un arbre est un ABR.
   Pour chaque noeud x, on doit vérifier que :
   - Les étiquettes de son sous-arbre gauche sont $\leq x$.
   - Les étiquettes de son sous-arbre droit sont $\geq x$.
   - Son sous-arbre gauche est un ABR.
   - Son sous-arbre droit est un ABR.

```
let est_abr a =
  let compare a b petit = match petit with
    (*Une fonction de comparaison qui permet de préciser si on veut <= ou >=*)
    |true -> a<=b
    |false -> a>=b
  in
  let rec compare_arbre a e petit =
    (*Si petit vaut false, vérifie si toute étiquette de a est plus grande que e.
    Si petit vaut true, vérifie si toute étiquette de a est plus petite que e*)
    match a with
    |Vide -> true
    |N(g,x,d) -> if not (compare x e petit) then false
                 else (compare_arbre g e petit) && (compare_arbre d e petit)
  in
  let rec aux a = match a with
    |Vide -> true
    |N(g,x,d) -> (compare_arbre g x true) && (compare_arbre d x false) && aux g && aux d
  in aux a;;
```

2 Arbres rouges-noirs

1. Construction

4. Comment rechercher une clé dans un arbre rouge-noir ?
   Comme on recherche dans un ABR.
5. Pour l'insertion, comment pourrait-on faire ? Pourquoi ajouter un noeud noir est une mauvaise idée ?
   On peut essayer d'insérer comme dans un ABR, mais il faut choisir la couleur pour notre nouveau noeud. S'il est noir, ça va forcément changer le nombre de noeuds noirs sur un chemin de la racine à un Vide, donc la troisième propriété ne sera plus vérifiée.

L'idée est donc d'ajouter une feuille rouge au bon endroit.

6. Trouver un exemple d'arbre rouge-noir tel qu'un ajout avec cette méthode fait qu'il ne vérifie plus la définition.
   Si en effectuant la recherche on tombe sur un Vide dont le père est rouge.
7. Quelle(s) propriété(s) des arbres rouges-noirs peuvent être violées par la méthode d'ajout proposée ?
   La première : si on ajoute dans l'arbre vide, la racine n'est pas noire.
   La troisième : cf question précédente.

Votre réponse à la question précédente devrait être 1 et 3. Si ce n'est pas le cas, réfléchissez encore.

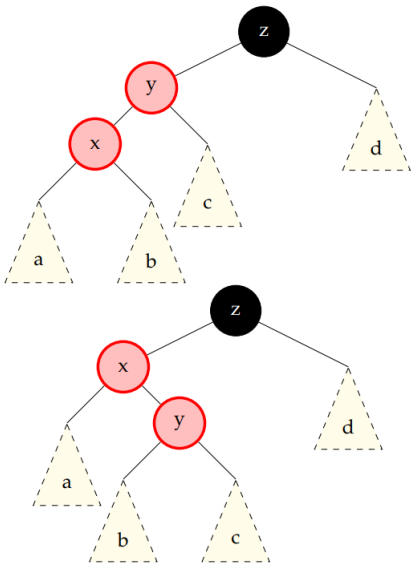
8. Si notre ajout viole la règle 1, comment corriger le problème de manière triviale ?
   On change la couleur.

On suppose maintenant que la règle violée est 3. (et pas 1.) : il existe un et un seul noeud rouge y ayant un fils rouge x. (Dans notre problème d'ajout intial, x est l'étiquette qu'on voulait rajouter)

9. Justifier que y a un père z et que celui-ci est noir.
   y est rouge, donc n'est pas la racine. y admet donc un père. Ce père est noir, car il a un fils rouge (y).

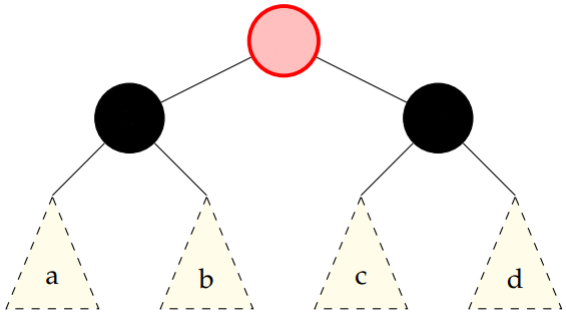
Il existe alors 4 configurations possibles selon si x est fils gauche ou droit de y et y est fils droit ou gauche de z.

Voici les configurations gg et gd. **Il est important de noter que la correction qu'on va apporter se fera récursivement en déplaçant le problème, d'où l'existence de sous-arbres pour** x



10. Dessiner les configurations dg et dd (les deux autres possibilités).
    Je vous laisse faire le dessin.

Pour rétablir les propriétés de rouge-noir dans l'arbre, on propose de transformer localement la partie posant problème en la configuration suivante :



11. Trouver comment on ramène les situations gg, gd, dg et dd à cette situation en effectuant des rotations gauches ou droites des arbres et en changeant les couleurs.
    Pour gg : On effectue un rotation droite sur y et z, puis on colorie x en noir.
    Pour gd : On effectue une rotation gauche, puis une rotation droite et on colorie x en noir.
    Pour dg : On effectue une rotation droite, puis une rotation gauche et on colorie z en noir.
    Pour dd : On effectue un rotation gauche sur y et z, puis on colorie z en noir.
12. Justifier que cette correction permet de conserver la dernière règle, et que la deuxième propriété est vérifiée **dans le sous-arbre obtenu après rotation**.
    Peu importe la configuration initiale, a, b, c et d ont même hauteur noire. Donc le sous-arbre originel et celui obtenu après correction ont même hauteur noire. Donc la hauteur noire a été préservée dans l'arbre total.
    Les sous-arbres a, b, c et d ne contiennent pas de noeud rouge avec un fils rouge, car notre méthode ne peut créer un problème qu'à un seul endroit. De plus dans la configuration corrigée, on voit qu'il n'y a pas de noeud rouge avec un fils rouge.
13. La première règle est-elle préservée ? Si non, comment corriger le problème ?
    Si z était la racine de l'arbre total, il est possible que la nouvelle racine soit rouge. Ce n'est pas grave, on la colorie en noir (comme c'est la racine, ça ne change pas la hauteur noire).

On a en revanche potentiellement crée un problème vis-à-vis de la règle 3. entre y et son nouveau père. On va donc itérer la correction.

14. Justifier que ce processus termine.
    La profondeur du noeud auquel le problème est diminue strictement au fur et à mesure de la correction.
15. En utilisant ce qu'on a vu, ajouter l'étiquette 3 dans l'arbre suivant en obtenant un arbre rouge-noir correct.
    Je vous laisse faire le dessin.

2. À propos de la hauteur

16. Soit un chemin entre un sommet s et une feuille f. Montrer qu'au moins la moitié des noeuds de ce chemin sont noirs.
    On raisonne par récurrence double sur la longueur de ce chemin, qu'on notera $s, c_1, \dots, c_m, f$.
    Si le chemin est de longueur 0, c'est à dire $s = f$, alors le chemin contient soit 1 rouge et 0 noir, soit 1 noir et 0 rouge. Dans les deux cas ce chemin vérifie la propriété. (quand on dit au moins la moitié, on parle en partie entière inférieure)
    Si le chemin est de longueur 1, c'est à dire qu'il ne contient que s et f, alors le chemin ne contient pas 2 rouges, c'est impossible d'après la 2ème propriété. Donc le chemin contient soit 1 noir 1 rouge, soit 2 noirs. Dans les deux cas ce chemin vérifie la propriété.
    Supposons la propriété vraie pour un chemin de longueur n. On va le montrer pour
17. Soient deux chemins $s, u_1, u_2, \dots, u_n$ et $s, c_1, c_2, \dots, c_m$ d'un sommet s à deux feuilles u_n et c_m. Montrer que le plus long de ces deux chemins a une longueur au plus égale au double de celle de l'autre chemin.
    D'après la propriété 3, les deux chemins contiennent $b(a)$ noeuds noirs. On note également nr_1 et nr_2 le nombre de noeuds rouges des deux chemins.
    D'après la question précédente, on a $b(a) \geq nr_1$ et $b(a) \geq nr_2$. On a aussi $nr_1 \geq 0$ et $nr_2 \geq 0$. Il est possible de n'avoir aucun noeud rouge dans un arbre rouge-noir, et par extension dans un chemin.
    La longueur du premier chemin est $b(a) + nr_1$. La longueur du deuxième chemin est $b(a) + nr_2$. Supposons que le premier chemin soit plus de deux fois plus long que le premier chemin.
    Alors $b(a) + nr_1 > 2(b(a) + nr_2)$, d'où $nr_1 > b(a) + 2nr_2$. Or $nr_2 \leq b(a)$, c'est impossible.
18. Montrer qu'un arbre rouge-noir à n noeuds a une hauteur au plus égale à $2\lfloor \log(n+1) \rfloor$.
    On considère un plus long chemin de la racine à une feuille f_1. Il est de longueur h.