1 Introduction

1.
```
let cons t g x d = match t with
  | V -> failwith "arbre vide"
  | N(_,_,_) -> N(g,x,d)
  | R(_,_,_) -> R(g,x,d)
;;
```

2 Insertion

2.
```
let corrige_rouge t = match t with
  | N (R (R (a, x, b), y, c), z, d)
  | N (R (a, x, R (b, y, c)), z, d)
  | N (a, x, R (R (b, y, c), z, d))
  | N (a, x, R (b, y, R (c, z, d)))
  | N (R (a, x, b), y, R (c, z, d))
    -> R (N (a, x, b), y, N (c, z, d))
  | t -> t
;;
```

3.
```
let rec insere_aux t x = match t with
  | V -> R (V, x, V)
  | R (l, y, r) | N (l, y, r) ->
    if x = y then t
    else if x < y then corrige_rouge (cons t (insere_aux l x) y r)
    else corrige_rouge (cons t l y (insere_aux r x))
;;
```

4.
```
let noircit t = match t with
  | R (g, x, d) -> N (g, x, d)
  | t -> t
;;

let insere t x = noircit (insere_aux t x) ;;
```

3 Suppression

3.1 Cas de base pour la suppression

5. On renvoie $\mathcal{A}$ dans les deux cas. Dans le premier cas, la hauteur noire du nouvel arbre est un de moins que celle de l'arbre initial, dans le deuxième cas c'est la même.

3.2 Suppression du minimum

6.
```
let rec supprime_min t = match t with
  | V -> failwith "vide"
  | R (V, x, d) -> d, false
  | N (V, x, d) -> d, true
  | R (g, x, d) | N (g, x, d) -> ...
;;
```

Remarque : on avait parlé de cas symétriques à la question précédente : ils ne sont pas présents ici, puisque l'on supprime le **minimum** et qu'on a donc toujours un fils **gauche** vide.

7. Les cercles rouges indiquent les nœuds susceptibles d'être rouges alors qu'ils ont un père rouge.
8. On suppose qu'on part d'un sous-arbre rouge-noir **correct**, et que le sous-arbre gauche aune hauteur noire strictement plus petite que celle du sous-arbre droit. En particulier, cela signifie que la hauteur noire du fils droit est non nulle.
   - Si la racine est rouge, les deux sous-arbres ont une racine noire, et le sous-arbre droit n'est pas vide : c'est notre premier cas.
   - Si la racine est noire :
     - soit le fils gauche est rouge (et donc non vide), ce qui correspond au deuxième cas ;
     - soit les deux fils sont noirs, et le fils droit non vide d'après la remarque ci-dessus : c'est le troisième cas ;
     - soit le fils gauche est noir et le fils droit est rouge. Les enfants du fils droit sont donc noirs, et comme la hauteur noire du fils droit est non nulle, ils ne peuvent pas être vides : c'est le quatrième cas.

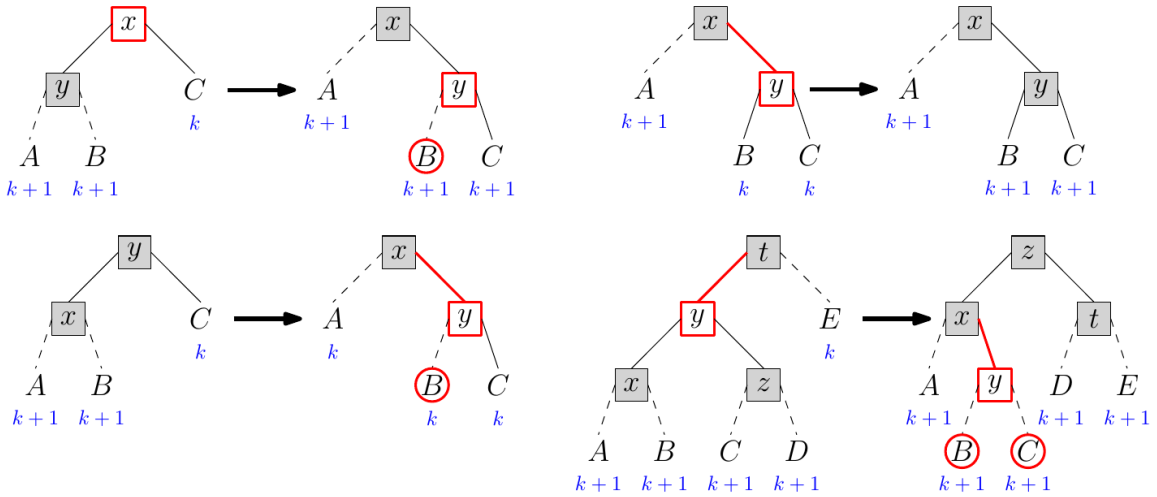
   Le fait que les cas soient correctement traités se vérifie facilement, surtout si l'on considère les annotations $k/k+1$ présentes sur les schémas. Les arbres obtenus en sortie vérifient la condition d'équilibre noir (dans le troisième cas, la hauteur noire a décru).

9.
```
let corrige_noir_gauche t b =
  if not b then (t, false)
  else match t with
    | R (a, x, N (b, y, c)) ->
      corrige_rouge (N (R (a, x, b), y, c)),
      false
    | N (R (a, x, b), y, c) ->
      N (N (a, x, b), y, c),
      false
    | N (a, x, N (b, y, c)) ->
      corrige_rouge (N (R (a, x, b), y, c)),
      true
    | N (a, x, R (N (b, y, c), z, N (d, t, e))) ->
      N (corrige_rouge (N (a, x, R (b, y, c))),
         z,
         N (d, t, e)),
      false
    | _ -> failwith "impossible"
;;
```

10.
```
let rec supprime_min t = match t with
  | V -> failwith "vide"
  | R (V, x, d) -> d, false
  | N (V, x, d) -> d, true
  | R (g, x, d) | N (g, x, d) ->
    let g', a_diminue = supprime_min g in
    corrige_noir_gauche (cons t g' x d) a_diminue
;;
```

3.3 Suppression d'un élément quelconque

11.



12.

```
let corrige_noir_droite t a_faire =
  if not a_faire then (t, false)
  else match t with
    | R (N (a, x, b), y, c) ->
      corrige_rouge (N (a, x, R (b, y, c))),
      false
    | N (a, x, R (b, y, c)) ->
      N (a, x, N (b, y, c)),
      false
    | N (N (a, x, b), y, c) ->
      corrige_rouge (N (a, x, R (b, y, c))),
      true
    | N (R (N (a, x, b), y, N (c, z, d)), t, e) ->
      N (corrige_rouge (N (a, x, R (b, y, c))),
         z,
         N (d, t, e)),
      false
    | _ -> failwith "impossible"
;;
```

13.
```
let rec minimum t = match t with
  | V -> failwith "arbre vide"
  | N(V,x,_)
  | R(V,x,_)
      -> x
  | N(g,_,_)
  | R(g,_,_)
      -> minimum g
;;

let rec supprime_aux t x = match t with
  | V -> V, false
  | N (g, y, d) | R (g, y, d) when x < y ->
    let g', a_diminue = supprime_aux g x in
    corrige_noir_gauche (cons t g' y d) a_diminue
  | N (g, y, d) | R (g, y, d) when x > y ->
    let d', a_diminue = supprime_aux d x in
    corrige_noir_droite (cons t g y d') a_diminue
  | N (V, _, t') | N (t', _, V) -> t', true
  | R (V, _, t') | R (t', _, V) -> t', false
  | N (g, _, d) | R (g, _, d) ->
    let m = minimum d in
    let d', a_diminue = supprime_min d in
    corrige_noir_droite (cons t g m d') a_diminue
;;
```

14.
```
let supprime t x =
  let t', _ = supprime_aux t x in
  noircit t'
;;
```