1.

```
let pere = function
  Vide -> failwith "pere inconnu"
 |Noeud n -> n.pere ;;

let fils_gauche = function
  Vide -> failwith "fils inconnu"
 |Noeud n -> n.gauche ;;

let fils_droit = function
  Vide -> failwith "fils inconnu"
 |Noeud n -> n.droit ;;

let valeur = function
  Vide -> failwith "arbre vide"
 |Noeud n -> n.val ;;

let couleur = function
  Vide -> Noir
 |Noeud n -> n.couleur ;;
```

2.

```
let est_fils_gauche a = fils_gauche (pere a)=a ;;

let est_fils_droit a = fils_droit (pere a)=a ;;
```

3.

```
let frere a =
  if est_fils_gauche a then  fils_droit (pere a)
  else fils_gauche (pere a) ;;

let oncle a = (frere (pere a)) ;;
```

4.

```
let rec transforme = function
   Vide -> V
 | Noeud n -> N (n.val,n.couleur,transforme n.gauche,transforme n.droit) ;;
```

5.

```
let rec test_pere_aux p = function
   Vide -> true
 | Noeud n -> n.pere=p && test_pere_aux (Noeud n) n.gauche
                       && test_pere_aux (Noeud n) n.droit;;

let test_pere = test_pere_aux Vide ;;
```

6.

```
let rec test_prop4 = function
  Vide -> true
 |Noeud n -> match n.couleur with
      Blanc -> (couleur n.pere)=Noir && test_prop4 n.gauche
                                     && test_prop4 n.droit
     |Noir  -> test_prop4 n.gauche && test_prop4 n.droit ;;
```

7.

```
exception StopProp5;;

let rec nb_noirs = function
  Vide -> 1
 |Noeud n -> let g = nb_noirs n.gauche and d= nb_noirs n.droit in
             if g=d then match n.couleur with
                   Blanc -> g
                  |Noir  -> g+1
             else raise StopProp5 ;;
```

Le déclenchement d'une exception avec la commande `raise` permet d'arrêter le calcul si à un nœud donné, la contrainte 5 n'est pas vérifiée.

8. La fonction `test_prop5` lance le calcul de `nb_noirs` et regarde si aucune exception n'est déclenchée. Si c'est le cas, la contrainte est vérifiée : le résultat est `true`. Dans le cas contraire l'exception `StopProp5` est détectée est le résultat est `false`.

```
let test_prop5 a =
  try
    nb_noirs a;
    true
  with
    StopProp5 -> false ;;
```

Pour se passer des exceptions il faudrait modifier la fonction `nb_noirs` pour qu'elle renvoie un couple formé d'un booléen qui indique si une violation de la contrainte 5 a été détectée et un entier qui renvoie la valeur calculée dans l'ancienne version.

9.

```
let testRN a =
  (couleur a)=Noir && test_prop4 a && test_prop5 a ;;
```

10.

```
let inverse_couleur = function
  Vide -> ()
 |Noeud n -> match n.couleur with
    Blanc -> n.couleur <- Noir
   |Noir  -> n.couleur <- Blanc ;;
```

11.

```
let adopte_gauche = fun
   Vide Vide -> failwith "arbres vides"
```

```
 | Vide (Noeud p) -> p.gauche <- Vide
 | (Noeud f) Vide -> f.pere <- Vide
 | (Noeud f) (Noeud p) -> ( p.gauche <- Noeud f;
                            f.pere <- Noeud p ) ;;

let adopte_droit f = fun
   Vide Vide -> failwith "arbres vides"
 | Vide (Noeud p) -> p.droit <- Vide
 | (Noeud f) Vide -> f.pere <- Vide
 | (Noeud f) (Noeud p) -> ( p.droit <- Noeud f;
                            f.pere <- Noeud p ) ;;
```

12.

```
let rotation_droite a =
  let b=fils_gauche a in
  let y=fils_droit b
  and p=pere a in
  if p=Vide || (est_fils_gauche a) then adopte_gauche b p
  else adopte_droit b p;
  adopte_gauche y a;
  adopte_droit a b ;;

let rotation_gauche b =
  let a=fils_droit b in
  let y=fils_gauche a
  and p=pere b in
  if p=Vide || (est_fils_gauche b) then adopte_gauche a p
  else adopte_droit a p;
  adopte_droit y b;
  adopte_gauche b a ;;
```

Ces deux fonctions sont beaucoups plus simple à écrire si on ne se soucie plus du champ `pere` des nœuds. C'est d'ailleurs le choix qui avait été fait dans l'épreuve d'informatique 2000 de Central dont est inspiré ce TP.

13.

```
let rotation_gauche_droite a =
  let b=fils_gauche a in
  rotation_gauche b;
  rotation_droite a ;;

let rotation_droite_gauche b =
  let a=fils_droit b in
  rotation_droite a;
  rotation_gauche b ;;
```

14.

```
exception FinInsertion ;;
```

```
let rec insere_blanc x = function
   Vide -> Noeud {val=x; couleur=Blanc; gauche=Vide; droit=Vide; pere=Vide}
 | Noeud n -> if n.val=x then raise FinInsertion
              else if x<n.val then match n.gauche with
                 Vide -> let new = Noeud {val=x; couleur=Blanc; gauche=Vide;
                                          droit=Vide; pere=Noeud n } in
                         (n.gauche <- new; new)
               | _ -> insere_blanc x n.gauche
              else match n.droit with
                 Vide -> let new = Noeud {val=x; couleur=Blanc; gauche=Vide;
                                          droit=Vide; pere=Noeud n } in
                         (n.droit <- new; new)
               | _ -> insere_blanc x n.droit
;;
```

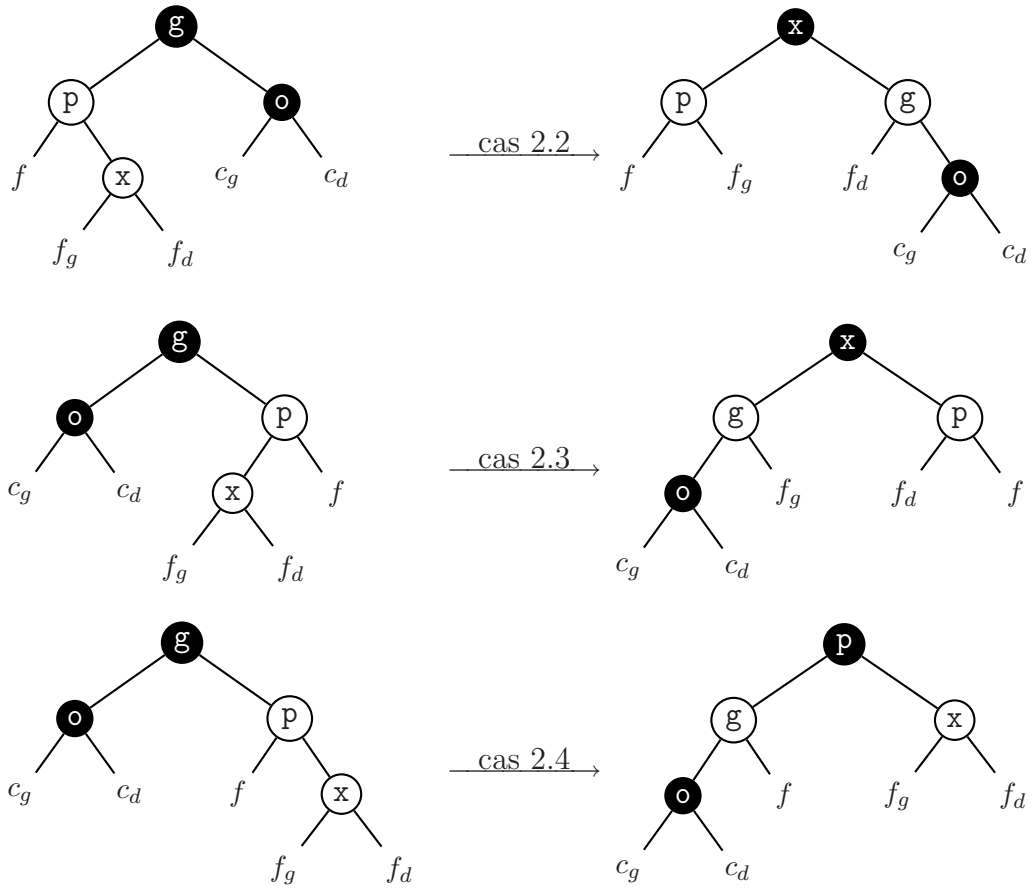
On peut là encore se passer de l'utilisation des exceptions avec un booléen qui indique si la valeur `x` est déjà présente dans l'arbre.

15.

```
let rec mauvais_blanc x =
   let p=pere x in
   match p with
     Vide     -> inverse_couleur x; (* x est la racine de l'arbre *)
   | Noeud np -> match np.couleur with
         Noir  -> ()
       | Blanc -> let o=frere p and g=pere p in
                  match couleur o with
                   Blanc -> ( inverse_couleur p;            (*  cas 1  *)
                              inverse_couleur o;
                              inverse_couleur g;
                              mauvais_blanc g  )
                  | Noir  -> if est_fils_gauche p then
                              if est_fils_gauche x then
                                ( rotation_droite g;        (* cas 2.1 *)
                                  inverse_couleur g;
                                  inverse_couleur p  )
                              else
                                ( rotation_gauche_droite g; (* cas 2.2 *)
                                  inverse_couleur g;
                                  inverse_couleur x  )
                             else
                              if est_fils_gauche x then
                                ( rotation_droite_gauche g; (* cas 2.3 *)
                                  inverse_couleur g;
                                  inverse_couleur x  )
                              else
                                ( rotation_gauche g;        (* cas 2.4 *)
```

```
                                    inverse_couleur g;
                                    inverse_couleur p ) ;;
```

Les trois cas manquants sont :



16. L'insertion d'une nouvelle valeur peut provoquer un changement de racine dans l'arbre. On modifie donc les fonctions précédentes pour envoyer la valeur de la racine :

```
let est_racine a = (pere a)=Vide ;;

let rec mauvais_blanc val_racine x =
   let p=pere x in
   match p with
     Vide     -> ( inverse_couleur x;  (* x est la racine de l'arbre *)
                   valeur x )
   | Noeud np -> match np.couleur with
       Noir  -> val_racine
     | Blanc -> let o=frere p and g=pere p in
                match couleur o with
                 Blanc -> ( inverse_couleur p;            (*  cas 1  *)
                            inverse_couleur o;
                            inverse_couleur g;
                            mauvais_blanc val_racine g   )
               | Noir  -> if est_fils_gauche p then
```

```
                        if est_fils_gauche x then
                          ( rotation_droite g;          (* cas 2.1 *)
                            inverse_couleur g;
                            inverse_couleur p;
                            if (est_racine p) then valeur p
                                              else val_racine )
                        else
                          ( rotation_gauche_droite g;  (* cas 2.2 *)
                            inverse_couleur g;
                            inverse_couleur x;
                            if (est_racine x) then valeur x
                                              else val_racine )
                      else
                        if est_fils_gauche x then
                          ( rotation_droite_gauche g;  (* cas 2.3 *)
                            inverse_couleur g;
                            inverse_couleur x;
                            if (est_racine x) then valeur x
                                              else val_racine )
                        else
                          ( rotation_gauche g;          (* cas 2.4 *)
                            inverse_couleur g;
                            inverse_couleur p;
                            if (est_racine p) then valeur p
                                              else val_racine ) ;;


let insere x a=
  try (mauvais_blanc (valeur a) (insere_blanc x a))
  with FinInsertion -> (valeur a) ;;
```