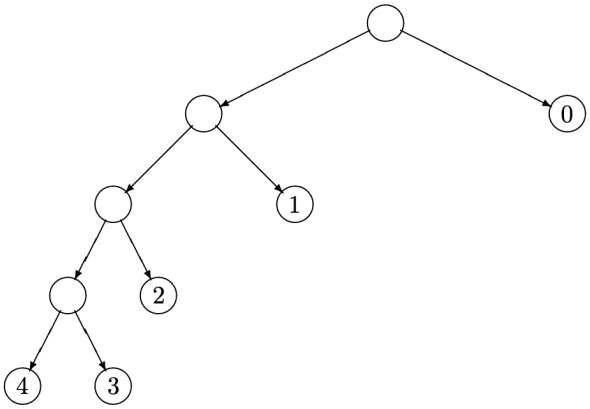
Correction du l'Interrogation 1 du Lundi 29 Septembre

Exercice 1 (Du cours)

Voir le cours...

Exercice 2 (Arbres peignes)

1. Un peigne rangé à cinq feuilles est par exemple :



2. Montrons par récurrence sur $n \geqslant 1$ que la hauteur d'un peigne rangé à n feuilles est égale à $n-1$.
   - Un peigne rangé à 1 feuille est une feuille et sa hauteur vaut 0.
   - Supposons le résultat vrai jusqu'à un rang $n \geqslant 1$. Un peigne rangé à $n+1$ feuilles s'écrit `Noeud(g,d)` avec `d` qui est une feuille et `g` qui est un peigne rangé à n feuilles. La hauteur de cet arbre est égale à 1 plus le maximum des hauteurs de `g` et `d` qui valent $n-1$ (hypothèse de récurrence) et 0. Notre arbre est donc de hauteur n, ce qu'il fallait prouver.

   Par le principe de récurrence, on a bien montré qu'un peigne rangé à n feuilles est de hauteur $n-1$.

3. Si l'arbre n'est pas une feuille, on vérifie qu'il y a une feuille à droite et, récursivement, que le fils gauche est un peine rangé.

```
let rec est_range a = match a with
  | Feuille x -> true
  | Noeud (g,Feuille x) -> est_range g
  | _ -> false
;;
```

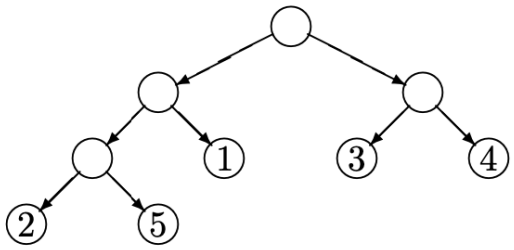
4. Si l'arbre n'est pas une feuille, on vérifie que l'un des fils est une feuille et, récursivement, que l'autre est un peigne strict.

```
let rec est_peigne_strict a = match a with
  | Feuille x -> true
  | Noeud(g,Feuille x) -> est_peigne_strict g
  | Noeud(Feuille x,d) -> est_peigne_strict d
  | _ -> false
;;
```

Il reste à particulariser la racine et à vérifier que, s'ils existent, fils gauche et droit sont des peignes stricts.

```
let est_peigne a = match a with
  | Feuille x -> true
  | Noeud (g,d) -> (est_peigne_strict g)&&(est_peigne_strict d)
;;
```

5. (a) Une rotation sur l'arbre exemple donne



(b) La rotation est possible dès que le fils droit est un noeud qui a au moins un fils qui est une feuille.

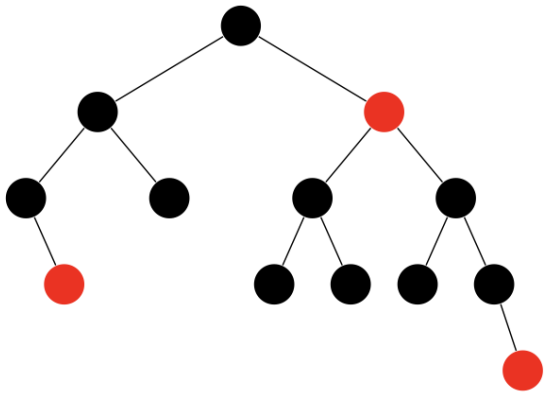
```
let rotation a = match a with
  | Noeud(a1,Noeud(a2,Feuille f))->Noeud(Noeud(a1,Feuille f),a2)
  | Noeud(a1,Noeud(Feuille f,a2))->Noeud(Noeud(a1,Feuille f),a2)
  | _ -> a
;;
```

(c) On écrit une fonction `range` qui agit comme `rangement` mais en supposant que l'argument est un peigne. Il reste alors à tester si c'est le cas.
L'idée de la fonction `range` est d'effectuer une rotation si c'est possible et de recommencer récursivement. On va finir par tomber sur le cas où le fils droit est une feuille. Il suffit alors de ranger le fils gauche.

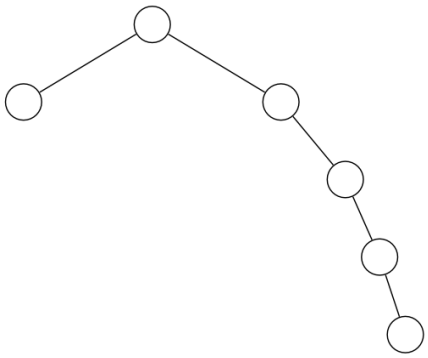
```
let rec range a = match a with
  | Feuille f -> Feuille f
  | Noeud(g,Feuille f) -> Noeud(range g,Feuille f)
  | _ -> range (rotation a)
;;

let rangement a =
    if est_peigne a then range a
    else a
;;
```

Exercice 3 (Arbres rouge-noir)

1. Le coloriage ci-dessous convient :



2. L'arbre ci-dessous ne peut pas avoir de coloration rouge-noir, car au moins trois noeuds du fils droit doivent être coloriés en rouge pour respecter la troisième condition, et ceci est impossible sans contrevenir à la deuxième condition.



3. (a) Il existe au moins un nœud à la profondeur $h(A)$. Le chemin qui le conduit à la racine comporte $h(A)+1$ nœuds dont $b(A)$ nœuds noirs et $h(A)+1-b(A)$ nœuds rouges. Ceci prouve déjà que $b(A) \leqslant h(A)+1$.

   Par ailleurs, sachant que la racine est noire et que tout parent d'un nœud rouge est noir, il y a aussi au moins $h(A)+1-b(A)$ nœuds noirs sur ce trajet, ce qui prouve l'inégalité : $h(A)+1-b(A) \leqslant b(A) \Leftrightarrow h(A)+1 \leqslant 2b(A)$.

   (b) Raisonnons par induction structurelle pour prouver cette inégalité.
   - Si A est vide, alors $b(A)=0$ et $|A|=0$.
   - Si $A=(x,F_g,F_d)$, A possède deux fils (éventuellement vides).

     Un fils noir ou vide F est la racine d'un arbre rouge-noir pour lequel $b(F)=b(A)-1$. Dans ce cas, $|F| \geqslant 2^{b(A)-1}-1$.

     Un fils rouge F ne peut avoir que des fils noirs. Il en a deux (éventuellement vides), qui sont les racines d'arbres rouge-noir A' pour lesquels $b(A')=b(A)-1$. Dans ce cas,

$$|F| \geqslant 2(2^{b(A)-1}-1)+1 = 2^{b(A)}-1.$$

     Sachant que $|A|=|F_g|+|F_d|+1$, on en déduit que

$$|A| \geqslant 2(2^{b(A)-1}-1) = 2^{b(A)}-1.$$

(c) Cette dernière inégalité peut aussi s'écrire $b(A) \leqslant \log(|A|+1)$ ce qui implique : $h(A) \leqslant 2\log(|A|+1) - 1$ et donc $h(A) = O(\log(|A|))$. Un arbre rouge-noir est bien équilibré.

4. (a)
```
let rec couleur_fils a = match a with
  | Vide -> true
  | N(N(_, Rouge, _), Rouge, _) -> false
  | N(_, Rouge, N(_, Rouge, _)) -> false
  | N(fg, _, fd) -> (couleur_fils fg) && (couleur_fils fd)
;;
```

(b) On construit une fonction auxiliaire qui renvoie un couple d'un booléen (si la condition sur la hauteur est respectée) et la valeur de la hauteur noire de l'arbre, puis une fonction chapeau qui retourne juste le booléen.

```
let rec aux a = match a with
  | Vide -> (true, 0)
  | N(Vide, Noir, fd) -> let (b, h) = aux fd in (b, h + 1)
  | N(fg, Noir, Vide) -> let (b, h) = aux fg in (b, h + 1)
  | N(fg, Noir, fd) -> let (b1, h1) = aux fg and (b2, h2) = aux fd in
                         (b1 && b2 && h1 = h2, h1 + 1)
  | N(Vide, Rouge, fd) -> aux fd
  | N(fg, Rouge, Vide) -> aux fg
  | N(fg, Rouge, fd) -> let (b1, h1) = aux fg and (b2, h2) = aux fd in
                          (b1 && b2 && h1 = h2, h1)
;;

  let hauteur_noire a = fst (aux a) ;;
```

(c) On vérifie les trois conditions pour que l'arbre soit rouge-noir.

```
let rouge_noir a = match a with
  | Vide -> false
  | N(_, c, _) -> (c = Noir) && (couleur_fils a) && (hauteur_noir a)
;;
```