Colle 12

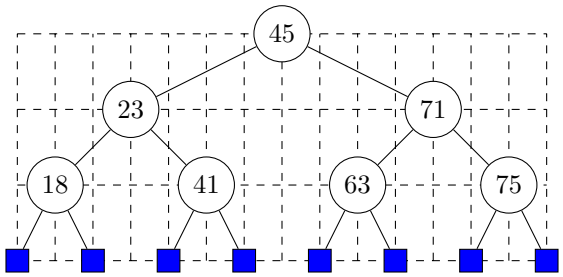
Dessiner des arbres

MPII : Ocaml

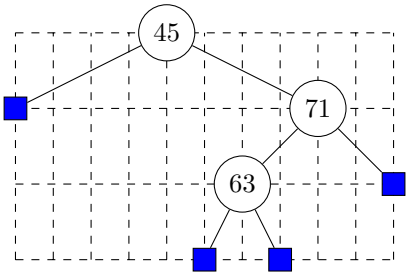
On se propose ici de représenter des arbres binaires, ce peut être utile pour visualiser les transformations que l'on effectue sur ces arbres.
On se donne deux contraintes :

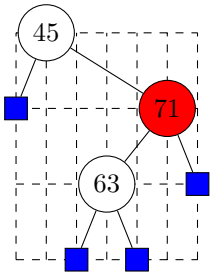
- les nœuds de même profondeur doivent alignés horizontalement,
- les nœuds des fils gauche (resp. droits) doivent être à gauche (resp. à droite) de la racine, cela induit que le parcours infixe pourra se faire en projetant verticalement.

Pour simplifier on placera les nœuds sur une grille régulière : la distance verticale entre un nœuds et ses fils est constante et la distance horizontale entre deux nœuds de même profondeur sera un multiple d'une longueur fixée.
Le cas d'un arbre complet ne donne pas lieu à beaucoup de choix si on veut éviter les irrégularités.



On notera qu'on a choisi de représenter les feuilles.
On peut se servir de ces positions pour tracer des arbres quelconques mais on peut préférer une représentation plus compacte.





C'est cette dernière forme que nous allons tracer : on veut un nœud ou une feuille à chaque position verticale. On ajoute une couleur aux nœuds, afin de pouvoir en mettre en évidence. Le type `color` est en fait un entier et les valeurs des couleurs seront définies par le module `graphics` introduit ensuite. Pour simplifier l'écriture, on ne considère que des valeurs entières aux nœuds.

```
type arbre = Vide | Noeud of arbre * int * color * arbre;;
```

L'exemple ci-dessus est codé par

```
let a0 = Noeud(Vide, 45, white,
               Noeud(Noeud(Vide, 63, white, Vide),
                     71, red, Vide));;
```

I Boites

I.1 Cadre de l'arbre

On peut maintenant débuter une représentation de l'arbre. On commencera par le cadre.
OCaml dispose d'une bibliothèque pour le tracé d'éléments simples : `graphics`.
Son utilisation dépend de la version de OCaml utilisée.

Versions antérieures à 4.08 La bibliothèque fait partie de l'installation. On la charge avec

```
#load "graphics.cma";;
```

Versions postérieures à 4.09 La bibliothèque n'est plus incluse. On doit la télécharger avec une instruction dans la console : `opam install graphics`. Pour en faciliter le chargement on utilisera `ocamlfind` : `opam install ocamlfind`. On peut alors charger la bibliothèque dans l'environnement OCaml :

```
#use "topfind";;
#require "graphics";;
```

On notera dans les deux cas l'écriture de `#` qui indique une directive et non une instruction.

Pour faciliter l'écriture des fonctions, on évite le préfixe `Graphics` par

```
open Graphics;;
```

On écrira ainsi `lineto 40 20;;` plutôt que `Graphics.lineto 40 20;;`.

Les instructions graphiques seront exécutées dans une fenêtre séparée que l'on doit ouvrir et fermer. Comme on souhaite avoir le temps de voir le contenu, on introduit une instruction qui attend un ordre (ici un clic dans la fenêtre) pour déclencher la fermeture.

```
open_graph " ";;
(* instructions graphiques *)
let _ = wait_next_event [Button_down] in close_graph ();;
```

Le nombre de commandes est important, pour la documentation on consultera la page
`https://ocaml.github.io/graphics/graphics/Graphics/index.html`
Il est important de noter que les dimensions et les angles (en degrés) sont des entiers.
La fenêtre graphique a des dimensions par défaut, on peut les modifier avec la souris ou par la fonction `resize_window`. Elle est repérée par des coordonnées cartésiennes x et y avec x position horizontale et y position verticale. L'origine (0, 0) est dans le coin inférieur gauche.

Nos dessins se feront du haut vers le bas (y décroissants) et de la gauche vers la droite. Les coordonnées du point supérieur gauche, les dimensions des pas et celles des éléments de l'arbre seront données par des variables dont on donne des exemples de valeurs.

```
let pas_h = 20;;
let pas_v = 50;;
let x0 = 20;;
let y0 = 400;;
let cote = 16
let rayon = 15;;
```

I.2 Dimensions

L'encombrement d'un arbre est donné sous forme d'entiers : les nombre de pas de base de la grille, horizontalement et verticalement. Par exemple l'arbre parfait est contenu dans une grille de 14 pas horizontaux et 3 pas verticaux, le second exemple dans une grille de 9 pas horizontaux et 3 pas verticaux. On remarque que le nombre de pas verticaux est la profondeur maximale d'une feuille et que le nombre de pas horizontaux est le nombre de nœuds et de feuille diminué de 1, c'est donc 2 fois le nombre de nœuds.

Question 1
Écrire une fonction `taille : 'a arbre -> int -> int` qui calcul la taille sous la forme d'un couple `(largeur, hauteur)`. On ne fera qu'une lecture de l'arbre

Pour dessiner un rectangle on a besoin des coordonnées du point inférieur gauche et des largeur et hauteur.

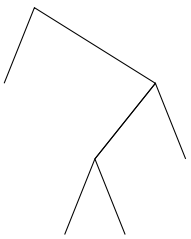
```
let rectangle arbre =
   let p, q = taille arbre in
   let larg = p*pas_h in
   let haut = q*pas_v in
   open_graph " 800x900";
   (* On peut imposer la taille de la fenêtre *)
   draw_rect x0 (y0 - haut) larg haut;
   let _ = wait_next_event [Button_down] in close_graph ();;
```

Question 2
Écrire une fonction fonction `rectangles : 'a arbre -> ()` qui dessine le rectangle encadrant chaque sous-arbre de l'arbre passé en paramètre.

On pourra écrire une fonction auxiliaire qui reçoit les coordonnées du point supérieur gauche d'un arbre comme paramètre et qui renvoie les coordonnées du points inférieur droit après avoir tracé le rectangle. Les arbres encadrant une feuille étant de taille nulle, il ne sera pas nécessaire de les dessiner.

II Tracé

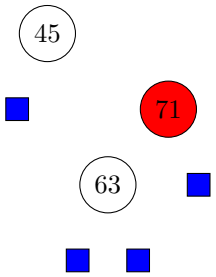
Question 3
Écrire une fonction `squelette : arbre -> unit` qui dessine tous les segments.
Une fonction auxiliaire pourra renvoyer aussi la position de la racine mais il n'est plus nécessaire de renvoyer une ordonnée.



On pourra utiliser les fonctions `moveto` et `lineto`.

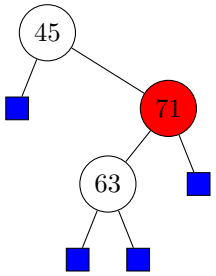
On va maintenant placer les nœuds et les feuilles. Un nœud sera représenté par un disque coloré par la couleur du nœud et de rayon donné par la variable `rayon` et une feuille par un carré bleu de longueur donnée par la variable `cote`, ces formes seront centrées aux extrémités des segments. La valeur d'un nœud sera écrite en noir dans le disque.

Question 4
Écrire une une fonction de dessin de la racine d'un arbre à la position (x, y).
`dessin_noeud : arbre -> int -> int -> unit`.



On pourra utiliser les fonctions `set_color`, `draw_rect`, `draw_circle`, `fill_rect`, `fill_circle`, `draw_string` et `string_of_int` qui convertit un entier en chaîne de caractères ainsi que les constantes `black`, `red`, `white` et `blue`.

Question 5
Écrire alors une fonction `noeuds : arbre -> unit` qui dessine tous les nœuds d'un arbre.

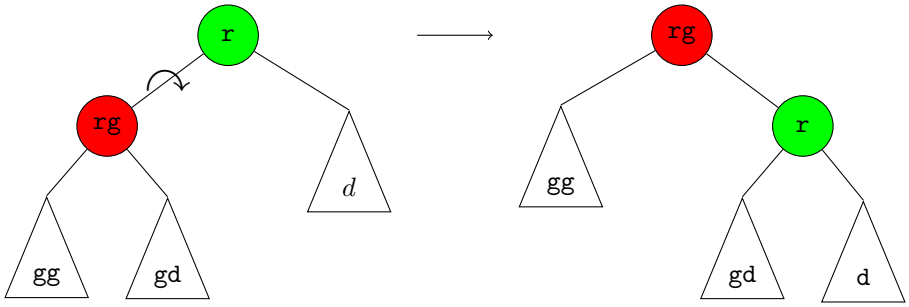


Pour dessiner le squelette et les nœuds, il faut faire attention à ne tracer les nœuds qu'après les segments.

Question 6
Écrire une fonction qui dessine l'arbre complet (squelette et nœuds) en ne faisant qu'une lecture de l'arbre.

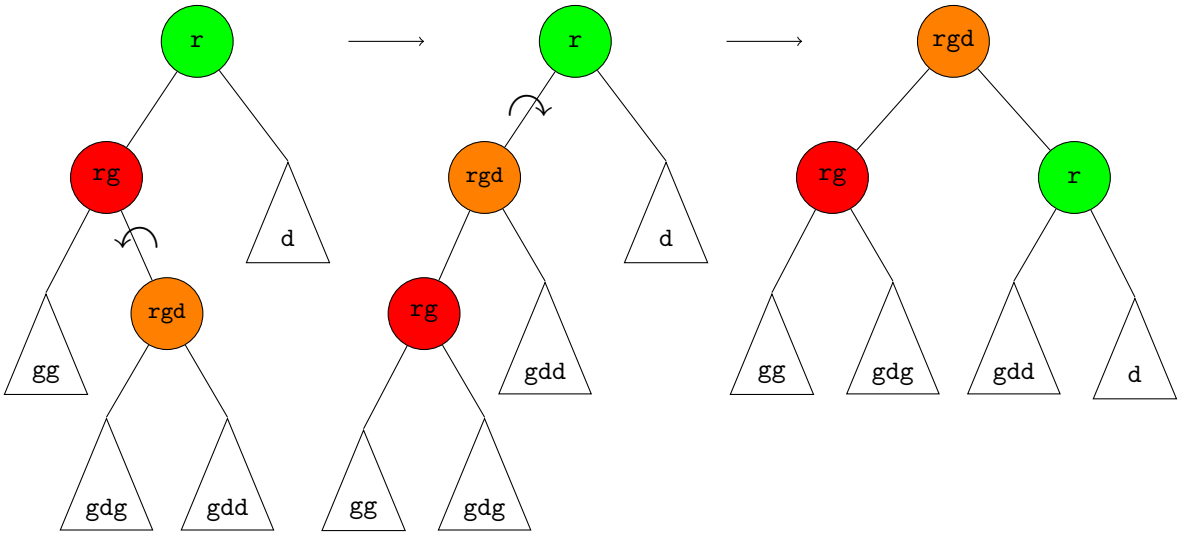
III Visualisation des rotations

Pour assurer une complexité logarithmique dans le traitement des arbres binaires on souhaitera équilibrer les arbres, c'est-à-dire déplacer des nœuds d'un coté à l'autre en conservant la structure sous-jacente, ici le parcours infixe doit être inchangé.
L'un des outils utilisés est la **rotation**. On représente ci-dessous la rotation droite, elle fait pivoter la liaison entre le nœud et son fils gauche dans le sens horaire : l'ancien fils gauche devient racine et l'ancienne racine devient son fils droit. On définit de même une rotation gauche.



Question 7
Écrire des fonctions `rotationD : 'a arbre -> 'a arbre` et `rotationG : 'a arbre -> 'a arbre` qui réalisent respectivement la rotation droite et la rotation gauche à la racine. Si la rotation n'est pas possible, on renverra le même arbre. Visualiser les effets.

Cette opération est adaptée quand c'est le sous-arbre `gg` qui apporte le déséquilibre à gauche, si c'est le sous-arbre `gd`, on voit qu'il n'a pas été remonté par la rotation. Dans ce cas on effectue une rotation gauche sur le fils gauche avant d'effectuer une rotation droite à la racine.



Question 8
Visualiser les effets de ces doubles rotations.

IV Visualisation des opérations des tas

Les tas sont implémentés sous forme de tableaux mais sont pensés sous la forme d'arbres binaires. Ici encore on simplifie les écriture avec des valeurs entières.

```
type heap = {mutable taille : int;
             mutable data : int array};;
```

Le tableau est mutable pour pouvoir augmenter la taille disponible en cas de dépassement.

```
let augmente tas =
  let n = tas.taille in
  let t = tas.data in
  let tt = Array.make (2*n) t.(0) in
  for i = 0 to (n-1) do tt.(i) <- t.(i) done;
  tas.data <- tt;;
```

On commence par l'écriture classique des fonctions de tas.

Question 9
Écrire les fonctions `creer_tas`, `extraire_max`, `inserer` et `supprimer_max`. On pourra s'inspirer du cours.

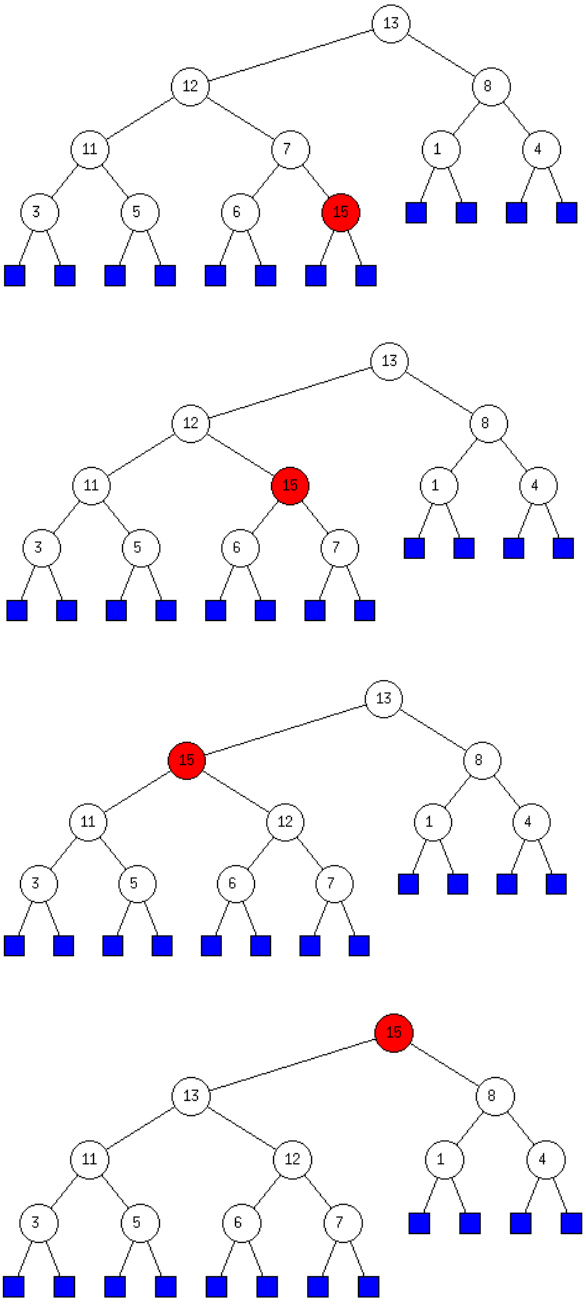
On a besoin de voir les tas, ce sera fait sous la forme d'un arbre avec la fonction `dessin` de la question **??**. La fonction doit donc transformer un tas en arbre puis le dessiner. La couleur des nœuds sera blanche à l'exception d'un nœud rouge dont l'indice sera un paramètre supplémentaire ;on donnera l'indice -1 si on ne souhaite pas mettre de nœud en évidence.

Question 10
Écrire une fonction `dessin_tas : heap -> int -> unit`.

Question 11
Modifier les fonctions `inserer` et `supprimer_max` pour qu'elles affichent les étapes des remontées et des descentes.

Voici ce que l'on peut obtenir avec l'insertion de 15 dans le tas

```
let t0 = {taille = 10;
          data = [|13; 12;  8; 11; 7; 1; 4; 3; 5; 6; 0; 0|]};;
```



Solutions

Solution de la question 1

Si les deux fils d'un arbre non vide ont des largeurs p et q l'arbre aura une largeur de $p+q+2$ car il faut placer la racine entre les deux fils. Si la rotation n'est pas possible, on renvoie le même arbre.

```
let rec taille arbre =
   match arbre with
   |Vide -> 0, 0
   |Noeud(g, r, c, d) -> let lg, hg = taille g
                         and ld, hd = taille d in
                         (lg + ld + 2, max hd hg + 1)
   |_ -> a;;
```

Solution de la question 2

```
let rectangles arbre =
   let rec aux x y a =
      match a with
      |Vide -> x, y
      |Noeud(g, r, c, d)->
         let x1, y1 = aux x (y - pas_v) g in
         let x2, y2 = aux (x1 + 2*pas_h) (y - pas_v) d in
         let yy = min y1 y2 in
         draw_rect x yy (x2 - x) (y - yy);
         x2, yy in
   open_graph " ";
   let _ = aux x0 y0 arbre in
   let _ = wait_next_event [Button_down] in close_graph ();;
```

Solution de la question 3

```
let squelette arbre =
   let rec aux x y a =
       match a with
       |Vide -> x, x
       |Noeud(g, r, c, d)->
           let x1, xr1 = aux x (y - pas_v) g in
           let x2, xr2 = aux (x1 + 2*pas_h) (y - pas_v) d in
           moveto xr1 (y -pas_v);
           lineto (x1 + pas_h) y;
           lineto xr2 (y -pas_v);
          x2, (x1 + pas_h) in
    open_graph " ";
    let _ = aux x0 y0 arbre in
    let _ = wait_next_event [Button_down] in close_graph ();;
```

Solution de la question 4

```
let dessin_noeud arbre x y =
  match arbre with
  |Vide ->  set_color blue;
            fill_rect (x-cote/2) (y-cote/2) cote cote;
            set_color black;
            draw_rect (x-cote/2) (y-cote/2) cote cote
  |Noeud(g, r, c, d) ->  set_color c;
                         fill_circle x y rayon;
                         set_color black;
                         draw_circle x y rayon;
                         moveto (x-5, y-5);
                         draw_string (string_of_int r);;
```

Solution de la question 5

```
let noeuds arbre =
   let rec aux x y a =
      match a with
      |Vide -> dessin_noeud a x y;
               x, x
      |Noeud(g, r, c, d)->
           let x1, xr1 = aux x (y - pas_v) g in
           dessin_noeud a (x1 + pas_h) y;
           let x2, xr2 = aux (x1 + 2*pas_h) (y - pas_v) d in
           x2, (x1 + pas_h) in
    open_graph " ";
    let _ = aux x0 y0 arbre in
    let _ = wait_next_event [Button_down] in close_graph ();;
```

Solution de la question 6

On ne trace pas les nœuds tout de suite

```
let dessin arbre =
   let rec aux x y a =
      match a with
      |Vide -> x, x
      |Noeud(g, r, c, d)->
           let x1, xr1 = aux x (y - pas_v) g in
           let x2, xr2 = aux (x1 + 2*pas_h) (y - pas_v) d in
           moveto xr1 (y -pas_v);
           lineto (x1 + pas_h) y;
           lineto xr2 (y -pas_v);
           dessin_noeud g xr1 (y - pas_v);
           dessin_noeud d xr2 (y - pas_v);
           x2, (x1 + pas_h) in
    open_graph " ";
    let x, xr = aux x0 y0 arbre in
    dessin_noeud arbre xr y0;
    let _ = wait_next_event [Button_down] in close_graph ();;
```

Solution de la question 7

```
let rotationD a =
   match a with
   |Noeud(Noeud(gg, rg, cg, gd), r, c, d)
      -> Noeud(gg, rg, cg, Noeud(gd, r, c, d))
   |_ -> a;;
```

```
let rotationG a =
   match a with
   |Noeud(g, r, c, Noeud(dg, rd, cd, dd))
      -> Noeud(Noeud(g, r, c, dg), rd, cd, dd)
   |_ -> a;;
```

Solution de la question 8

Solution de la question 9

```
let creer_tas () =
   {taille = 0 ; data = Array.make 2 0};;
```

```
let echange t i j =
   let temp = t.(i) in
   t.(i) <- t.(j);
   t.(j) <- temp;;
```

```
let rec remonter i tas =
   let t = tas.data in
   if i > 0
   then let j = (i-1)/2 in
      if t.(j) < t.(i)
      then begin echange t i j;
           remonter j tas end ;;
```

```
let inserer x tas =
   let n = tas.taille in
   if n = Array.length tas.data then augmente tas;
   tas.data.(n) <- x;
   tas.taille <- n + 1;
   remonter n tas;;
```

```
let extraire_max tas =
   if tas.taille = 0
   then failwith "Le tas est vide";;
   else tas.data.(0);;
```

```
let filsGrand k tas =
   let n = tas.taille in
   let t = tas.data in
   if n <= 2*k + 1
   then None
   else if n = 2*k + 2
        then  Some (2*k + 1)
        else if  t.(2*k+1) > t.(2*k+2)
             then Some (2*k + 2) else Some (2*k + 1);;
```

```
let rec descendre k tas =
   let t = tas.data in
   match filsGrand k tas with
   |Some p when t.(p) > t.(k)
      -> echange t k p;
         descendre p tas
   |_ -> ();;
```

```
let enlever_max tas =
   let n = tas.taille -1 in
   let t = tas.data in
   echange t 0 n;
   tas.taille <- n;
   descendre 0 tas;;
```

Solution de la question 10

```
let dessin_tas tas i0=
   let n = tas.taille in
   let t = tas.data in
   let rec faire i =
      if i >= n
      then Vide
      else let c = if i = i0 then red else white in
           Noeud(faire (2*i+1), t.(i), c, faire (2*i+2)) in
   dessin (faire 0);;
```

Solution de la question 11

```
let rec remonter i tas =
   let t = tas.data in
   if i > 0
   then let j = (i-1)/2 in
      if t.(j) < t.(i)
      then begin echange t i j;
           dessin_tas tas j;
           remonter j tas end ;;
```

```
let inserer x tas =
   let n = tas.taille in
   if n = Array.length tas.data then augmente tas;
   tas.data.(n) <- x;
   tas.taille <- n + 1;
   dessin_tas tas n;
   remonter n tas;;
```

```
let rec descendre k tas =
   let t = tas.data in
   match filsGrand k tas with
   |Some p when t.(p) > t.(k)
      -> dessin_tas tas k;
         echange t k p;
         descendre p tas
   |_ -> dessin_tas tas k;;
```

```
let enlever_max tas =
   let n = tas.taille -1 in
   let t = tas.data in
   dessin_tas tas 0;
   echange t 0 n;
   tas.taille <- n;
   descendre 0 tas;;
```