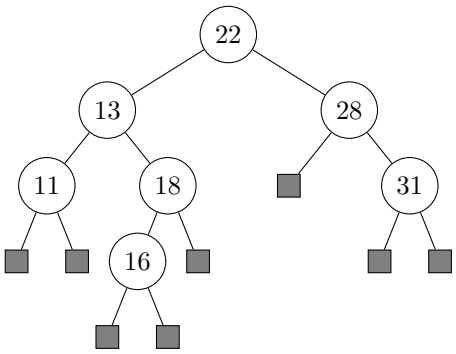
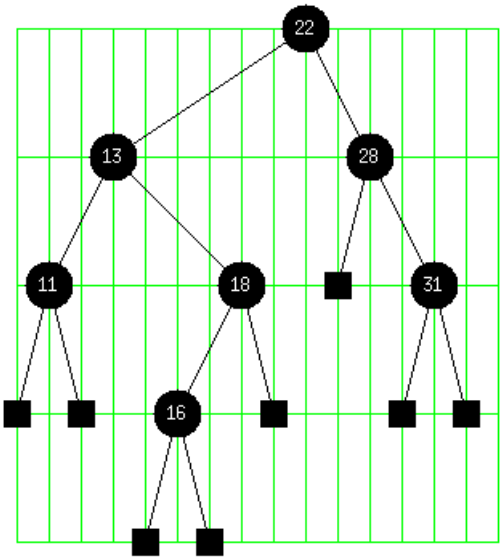
T.P. A

I – Tracé d'arbres binaires

On se propose ici de représenter des arbres binaires afin de pouvoir visualiser les opérations que l'on effectue sur ces arbres. On souhaite dessiner les arbres de telle manière que le parcours infixe consiste à lire les nœuds (et les feuilles) de gauche à droite ; on impose aussi que les nœuds de même profondeur soient alignés horizontalement. On placera donc les nœuds sur une grille.

L'objectif est de faire tracer à OCaml l'arbre (abstrait) de gauche sous la forme de droite





Nous allons travailler par renforcements successifs d'une fonction : dans chaque partie elle sera améliorée.
Le type de donnée est celle du cours pour des arbres binaires d'entiers.

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

1 Rectangles

Dans cette partie nous allons encadrer chaque sous-arbre dans un rectangle.

Dans un premier temps on va calculer la taille de la grille qui contient l'arbre. On a besoin

- du nombre de pas verticaux, c'est la profondeur maximale des feuilles donc la hauteur augmentée de 1,
- du nombre de pas horizontaux, comme on dessine un nœud ou une feuille par pas, c'est le nombre total de nœuds/feuilles diminué de 1 donc le double du nombre de nœuds. (Rappel : le nombre de feuilles est le nombre de nœud plus 1.) Plus simplement 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.

Question 1.1 (Solution page 16)
Écrire une fonction qui calcul la taille sous la forme d'un couple (largeur, hauteur).
On ne fera qu'une lecture de l'arbre

```
taille : arbre -> int * int
```

On va maintenant dessiner le cadre d'un arbre.

Pour faire des représentations graphique on a besoin de la bibliothèque graphique.

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

1. La première ligne charge en mémoire le fichier de la bibliothèque.
2. La deuxième ligne permet d'utiliser les fonctions de la bibliothèque directement : on écrit draw au lieu de Graphics.draw

Une documentation complète se trouve à

https://ocaml.github.io/graphics/graphics/Graphics/index.html

- Les dessins se font dans une fenêtre que l'on doit ouvrir puis fermer.

  ```
  open_graph " 800x900";;
  ...
  close\_graph ();;
  ```

  Ici on a ouvert une fenêtre graphique de 800 pixels de large et 900 pixels de haut.
  L'espace blanc avant le premier chiffre est obligatoire.
- Les coordonnées sont calculées en pixels (ce sont des entiers),
  l'origine est en bas à gauche.
- Pour observer le dessin avant de le fermer, on pourra utiliser l'instruction

  ```
  let _ = wait_next_event [Button_down] in close_graph();
  ```

  Elle met le programme en pause en attendant un clic de souris.
- On définit les constantes

  ```
  let pas_h = 20;;
  let pas_v = 80;;
  let x0 = 20;;
  let y0 = 800;;
  ```

  pas_h et pas_v sont les tailles, respectivement horizontale et verticale, de la grille.
  x_0 et y_0 sont les coordonnées du point de départ qui correspond au point en haut et à gauche du rectangle, en effet l'arbre sera dessiné depuis sa racine.
- On peut dessiner un rectangle avec la fonction draw_rect x y larg haut : x et y sont les coordonnées du point en bas à droite, larg est la largeur, larg est la hauteur.

Question 1.2 (Solution page 8)
Écrire une fonction rectg x1 y1 x2 y2 qui dessine un rectangle à partir des coordonnées des points supérieur gauche et inférieur droit.

```
rectg : int -> int -> int -> int -> unit
```

Question 1.3 (Solution page 8)
Écrire une fonction bas_droite a b arbre qui renvoie les coordonnées du point en bas à droite du rectangle qui délimite l'arbre et dont le point en haut à gauche est de coordonnées a et b.

```
bas_droite : int -> int -> arbre -> int * int
```

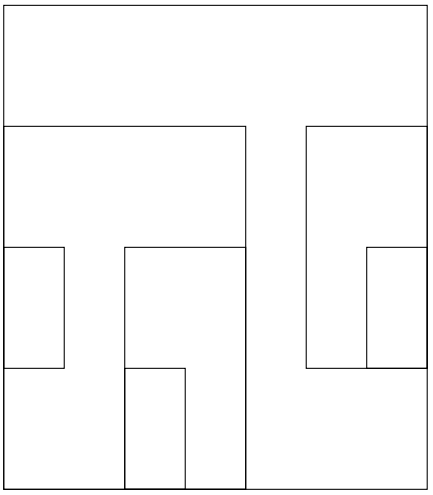
Voici une fonction qui dessine le rectangle délimitant un arbre.

```
let rectangle arbre =
  let x2, y2 = bas_droite x0 y0 arbre in
  open_graph " 800x900";
  rectg x0 y0 x2 y2;
  let _ = wait_next_event [Button_down] in close_graph ();;
```

Question 1.4 (Solution page 8)
Écrire une fonction rectangles arbre qui dessine tous les rectangles délimitant les sous-arbres non réduits à une feuille.

```
rectangles : arbre -> unit
```

La fonction devra faire appel à une fonction récursive adaptée de bas_droite, afin de pouvoir encadrer la fonction récursive par l'ouverture et la fermeture de la fenêtre. Cette fonction récursive aura comme paramètres les coordonnées du coin supérieur gauche du rectangle qui délimite l'arbre, pour l'arbre global ces coordonnées sont x_0 et y_0.

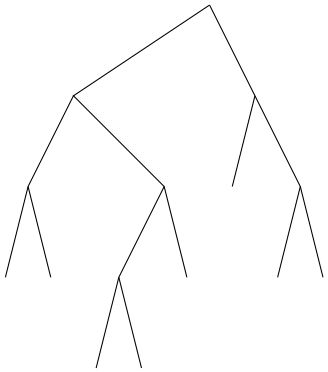


2 Tracé

2.1 Squelette

On souhaite maintenant tracer le squelette de l'arbre, c'est-à-dire l'ensemble des segments entre les nœuds et leurs fils (vides ou non).
Pour dessiner des segments on utilise 2 fonctions de déplacement :

- moveto x y déplace la position sans rien tracer jusqu'à (x, y)
- lineto x y trace un trait depuis la position actuelle jusqu'a (x, y) .

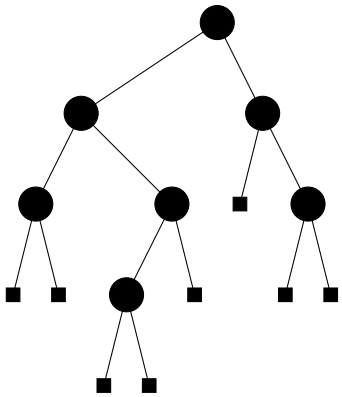


Question 2.1 (Solution page 16)
Écrire une fonction squelette arbre qui dessine tous les segments.
Une fonction auxiliaire pourra renvoyer aussi la position de la racine.

```
squelette : arbre -> unit
```

2.2 Nœuds

On va maintenant placer les nœuds et les feuilles. Un nœud sera représenté par un disque de rayon 15 (par exemple) et une feuille par un carré de longueur 16, ces formes seront centrées aux extrémités des segments.



```
let cote = 16
let rayon = 15;;
```

Question 2.2 (Solution page 17)
Écrire une une fonction de dessin de la racine d'un arbre à la position (x, y).

```
dessinNoued arbre -> int -> int -> unit
```

Question 2.3 (Solution page 9)
Écrire une fonction seg_noeuds arbre qui dessine les segments et les nœuds.

```
seg_noeuds: arbre -> unit
```

2.3 Valeur des nœuds

Il reste alors à écrire la valeur du nœud dans le cercle.
On utilise les fonctions

- draw_string texte qui écrit un texte avec la couleur courante depuis la position actuelle (elle correspond au point en bas à gauche).
- string_of_int qui convertit un entier en chaîne de caractères.

Question 2.4 (Solution page 9)
Écrire une fonction dessin arbre qui dessine les segments et les nœuds ainsi que les valeurs, supposées entières, des nœuds.

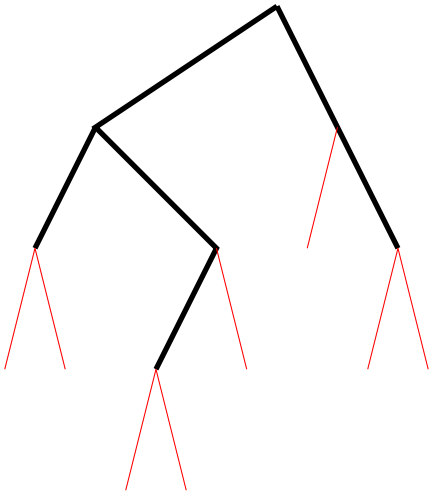
```
dessin : arbre -> unit
```

3 Décoration

3.1 Squelette

On peut décorer les lignes du squelette (ces fonctions agissent aussi sur les rectangles) :

- set_color sélectionne la couleur, les couleurs de base sont pré-définies : black, while, red, green, blue, yellow, cyan, magenta.
- set_line_width sélectionne la largeur du trait.

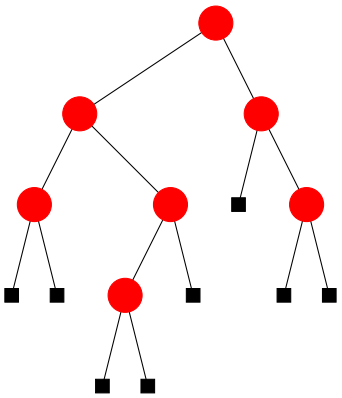


Question 3.1 (Solution page 17)
Écrire une fonction squelette2 arbre qui dessine les segments.
Les segments vers des nœuds non vides seront plus épais, ceux vers une feuille seront rouge.

```
squelette : arbre -> unit
```

3.2 Noeuds

On va parfois dessiner les nœuds en couleur.



Question 3.2 (Solution page 18)
Écrire une fonction seg_noeudsRN arbre qui dessine les segments et les nœuds avec les nœuds en rouge et les feuilles en noir.

```
seg_noeudsRN : arbre -> unit
```

Si on n'y prend pas garde les segment sont dessinés au-dessus des nœuds, il faudra adapter le dessin.

4 Solutions

Correction de la question 1.1 page 13
Comme la largeur est le nombre de nœuds/feuilles moins un, on doit ajouter 2 lors de la construction.

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

Correction de la question 1.2 page 5

```
let rectg x1 y1 x2 y2 =
draw_rect x1 y2 (x2 - x1) (y1 - y2);;
```

Correction de la question 1.3 page 5

```
let rec bas_droite a b arbre =
  match arbre with
  |Vide -> a, b
  |Noeud(g, r, d)->
            let xg, yg = bas_droite a (b - pas_v) g in
            let xd, yd = bas_droite (xg + 2*pas_h)
                                    (b - pas_v) d in
  xd, min yg yd;;
```

Correction de la question 1.4 page 5

```
let rectangles arbre =
    let rec aux x y a =
        match a with
        |Vide -> x, y
        |Noeud(g, r, 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
                rectg x y x2 yy;
                x2, yy in
    open_graph " 800x900";
    let _ = aux x0 y0 arbre in
    let _ = wait_next_event [Button_down] in close_graph ();;
```

Correction de la question 2.1 page 14

```
let squelette arbre =
    let rec aux x y a =
        match a with
        |Vide -> x, y, x
        |Noeud(g, r, d)->
                let x1, y1, xr1 = aux x (y - pas_v) g in
                let x2, y2, 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, (min y1 y2), (x1 + pas_h) in
    open_graph " 800x900";
    let _ = aux x0 y0 arbre in
    let _ = wait_next_event [Button_down] in close_graph ();;
```

Correction de la question 2.2 page 14

```
let dessinNoeud arbre x y =
  match arbre with
  |Vide ->  fill_rect (x-cote/2) (y-cote/2) cote cote
  |Noeud(g, r, d) ->  fill_circle x y rayon;;
```

Correction de la question 2.3 page 6

```
let seg_noeuds arbre =
    let rec aux x y a =
        match a with
        |Vide -> dessinNoeud a x y;
                 x, y, x
        |Noeud(g, r, d)->
                let y0 = y - pas_v in
                let x1, y1, xr1 = aux x 0 g in
                let x2, y2, xr2 = aux (x1 + 2*pas_h) y0 d in
                let xr = x1 + pas_h in
                moveto xr1 y0;
                lineto xr y;
                lineto xr2 y0;
                dessinNoeud a xr y;
                 x2, (min y1 y2), xr in
    open_graph " 800x900";
    let _ = aux x0 y0 arbre in
    let _ = wait_next_event [Button_down] in  close_graph ();;
```

Correction de la question 2.4 page 6
On commence par modifier le dessin des nœuds pour inclure le texte.

```
let toutNoeud_r arbre x y =
    match arbre with
    |Vide ->  fill_rect (x-cote/2) (y-cote/2) cote cote
    |Noeud(g, r, d) ->  fill_circle x y rayon;
                        set_color white;
                        let ch = string_of_int r in
                        let k = String.length ch in
                        moveto (x - 3*k + 1) (y - 5);
                        draw_string ch;
                        set_color black;;
```

Ensuite on dessine les nœuds des fils au lieu de celui de la racine dans la fonction, il faut alors dessiner la racine à la fin.

```
let dessin arbre =
    let rec aux x y a =
        match a with
        |Vide -> x, y, x
        |Noeud(g, r, d)->
                let y0 = y - pas_v in
                let x1, y1, xr1 = aux x 0 g in
                let x2, y2, xr2 = aux (x1 + 2*pas_h) y0 d in
                let xr = x1 + pas_h in
                moveto xr1 y0;
                lineto xr y;
                lineto xr2 y0;
                toutNoeud_r g xr1 y0;
                toutNoeud_r d xr2 y0;
                 x2, (min y1 y2), xr in
    open_graph " 800x900";
    let a, b, xr = aux x0 y0 arbre in toutNoeud_r arbre xr y0;
    let _ = wait_next_event [Button_down] in  close_graph ();;
```

Correction de la question 3.1 page 15
Il est utile d'écrire une fonction séparée pour tracer un segment selon sa destination.

```
let segment x1 y1 x2 y2 a =
    match a with
    |Vide -> moveto x1 y1;
             set_color red;
             set_line_width 1;
             lineto x2 y2
    |_ -> moveto x1 y1;
          set_color black;
          set_line_width 2;
          lineto x2 y2;;
```

```
let squelette2 arbre =
    let rec aux x y a =
        match a with
        |Vide -> x, y, x
        |Noeud(g, r, d)->
                let y0 = y - pas_v in
                let x1, y1, xr1 = aux x y0 g in
                let x2, y2, xr2 = aux (x1 + 2*pas_h) y0 d in
                let xr = x1 + pas_h in
                segment xr y xr1 y0 g;
                segment xr y xr2 y0 d;
                dessinNoeud a xr y;
                x2, (min y1 y2), xr in
    open_graph " 800x900";
    let _ = aux x0 y0 arbre in
    let _ = wait_next_event [Button_down] in  close_graph ();;
```

Correction de la question 3.2 page 15
On commence par modifier le dessin des nœuds.

```
let dessinNoeudRN arbre x y =
    match arbre with
    |Vide ->  fill_rect (x-cote/2) (y-cote/2) cote cote
    |Noeud(g, r, d) ->  set_color red;
                        fill_circle x y rayon;
                        set_color black;;
```

Ensuite on dessine les nœuds des fils au lieu de celui de la racine dans la fonction, il faut alors dessiner la racine à la fin.

```
let seg_noeudsRN arbre =
    let rec aux x y a =
        match a with
        |Vide -> x, y, x
        |Noeud(g, r, d)->
                let y0 = y - pas_v in
                let x1, y1, xr1 = aux x 0 g in
                let x2, y2, xr2 = aux (x1 + 2*pas_h) y0 d in
                let xr = x1 + pas_h in
                moveto xr1 y0;
                lineto xr y;
                lineto xr2 y0;
                dessinNoeudRN g xr1 y0;
                dessinNoeudRN d xr2 y0;
                 x2, (min y1 y2), xr in
    open_graph " 800x900";
    let a, b, xr = aux x0 y0 arbre
      in dessinNoeudRN arbre xr y0;
    let _ = wait_next_event [Button_down] in close_graph ();;
```