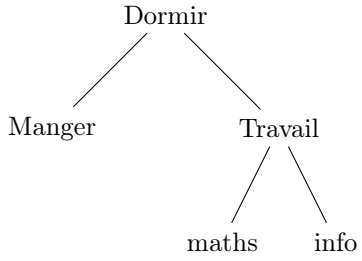
▶ **MP – Option Informatique**
3ème TP Caml
sl@jachiet.com
http://jachiet.com/tps
Lundi 12 novembre 2012

Arbres

Dans ce tp, on va manipuler des arbres où chaque nœud peut contenir une valeur. On définit le type suivant :

```
type 'a tree = Empty | Node of ('a tree) * 'a *('a tree)
```

De cette manière l'arbre :



sera représenté par :

```
type 'a tree = Empty | Node of ('a tree) * 'a *('a tree)
let vie = Node (
                Node (Empty, "manger", Empty),
                "dormir",
                Node (
                        Node (Empty, "maths", Empty),
                        "Travail",
                        Node (Empty, "info", Empty)
                        )
                )
```

1 Échauffement

▶ **Question 1** *Programmer une fonction qui renvoie le nombre de nœuds dans un arbre. Faire la même pour le nombre de feuilles.*

▶ **Question 2** *Programmer une fonction qui renvoie la hauteur d'un arbre.*

▶ **Question 3** *Programmer une fonction qui renvoie la liste des étiquettes d'un arbre dans l'ordre infixe (i.e. d'abord celles du sous-arbre gauche, puis l'étiquette du nœud puis celles du sous-arbre droit). La complexité attendue est $O(n)$.*

2 Arbre binaire de recherche

Un arbre binaire de recherche est un arbre qui satisfait la propriété (par rapport à une relation d'ordre donnée) : toutes les étiquettes des fils gauches d'un nœud sont plus petites que celle du nœud courant et celles des fils droits sont plus grandes.
Pour chacune des questions suivantes vous devez calculer la complexité et comparer avec la complexité si l'arbre n'était pas un arbre binaire de recherche

▶ **Question 4** *Programmer une fonction qui prend un arbre t et une étiquette x et détermine si x est une étiquette dans t. Quelle est la complexité ? Quelle serait la complexité si l'arbre n'était pas ordonné ?*

▶ **Question 5** *Programmer une fonction qui prend un ABR t et une étiquette x et renvoie un ABR qui contient exactement les étiquettes de t et x.*

▶ **Question 6** *Déduire de la question précédente un algorithme de tri. Quel est le pire cas ? Le meilleur ? Que se passe-t-il, en moyenne, si l'on mélange la liste avant de la trier ?*

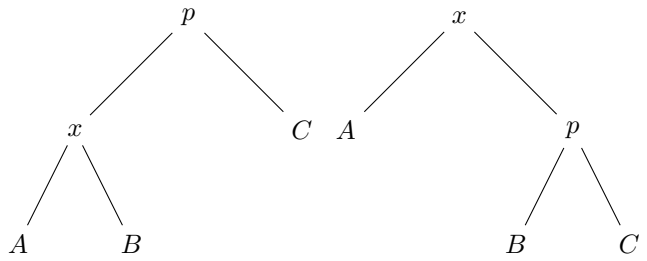
▶ **Question 7** *Déterminer (empiriquement) la hauteur en fonction du nombre d'éléments en insérant des entiers choisis aléatoirement (entre 0 et 10^8 par exemple). Rappel : Utiliser la fonction random__int.*

▶ **Question 8** *Programmer une fonction qui prend deux arbres et renvoie leur union. Ne cherchez pas trop compliqué.*

▶ **Question 9** *Écrire la fonction qui supprime un élément dans un ABR. Il est recommandé d'écrire une fonction qui extrait le min (ou le max) d'un sous arbre pour commencer.*

3 Rotation et Treaps

Le problème avec les ABR, c'est qu'ils peuvent être déséquilibrés, c'est-à-dire que leur hauteur est sensiblement plus grande que $log_2(n)$. Pour fabriquer un ABR très déséquilibré, il suffit d'insérer des entiers en ordre croissant. Pour faire face à cette situation, on utilise des procédures d'équilibrage. Ces procédures font généralement largement appel aux rotations (cf. dessins plus bas). S'il y a beaucoup plus de noeuds à droite qu'à gauche de la racine, on peut en faire passer un peu de l'autre côté de la sorte.



▶ **Question 10** *Programmer deux fonctions qui réalisent la rotation gauche et la rotation droite. Elles déclenchent une exception si la rotation n'est pas possible.*
Un tas (heap en anglais) est un arbre où les étiquettes des nœuds satisfont une relation moins contraignante que sur les ABR. En effet, dans un heap, les étiquettes des fils doivent être plus grandes que celle de leur père. Un Treap (Tree+Heap) est une structure de donnée conçue pour éviter le pire cas des ABR. Chaque noeud contient une étiquette (comme dans les ABR) et une priorité, qui est un entier tiré au hasard lors de l'insertion du nœud. On veut que les étiquettes forment un ABR, et les priorités un Heap.

▶ **Question 11** *Définir un type pour les Treaps.* `Astuce : Ré-utiliser celui des arbres.`

▶ **Question 12** *Programmer une fonction qui teste l'appartenance d'une étiquette à un Treap.*

▶ **Question 13** *Écrire une fonctio qui étant donné une étiquette et un nœud vérifie que l'étiquette pourrait être du père du nœud donné en respectant la propriété de tas.*

▶ **Question 14** *Écrire une fonction qui prend en paramètre un treap où seul un des fils direct de la racine ne respecte pas la propriété de tas et le corrige de manière adéquate.*

▶ **Question 15** *Programmer la fonction insertion. L'insertion se passe de la manière suivante : on détermine sous quelle feuille l'étiquette doit être insérée (comme dans un ABR), on tire ensuite une priorité au hasard pour le nouveau nœud et tamise le tas, c'est à dire qu'on fait remonter (par des rotations) le nœud jusqu'à ce que sa priorité soit plus grande que celle de son père.*

▶ **Question 16** *Programmer la suppression d'un nœud. Attention aux invariants !*

▶ **Question 17** *Écrire une fonction* `split x t` *qui étant donné une valeur x et un treap t deux Treaps $t_{\leq}$ et $t_{>}$ contenant respectivement les étiquettes inférieurs et supérieures à x. Indice : utilisée astucieusement, la fonction* `insertion` *fait déjà tout le travail.*

▶ **Question 18** *Programmer l'union et l'intersection de deux Treaps. Penser à utiliser* `split` *astucieusement.*

▶ **MP – Option Informatique**
3ème TP Caml
sl@jachiet.com
http://jachiet.com/tps
Lundi 12 novembre 2012

Arbres

Un corrigé

▶ Question 1

```
let rec noeuds = function
  | Node(a,_,b) -> 1 + (noeuds a) + (noeuds b)
  | Empty -> 0
;;
let feuilles = function
| Node(a,b) -> max 1 ((feuilles a) + (feuilles b))
| Empty -> 0
;;
```

▶ Question 2

```
let rec hauteur = function
| Empty -> 0
| Node(a,b) -> max (hauteur a) (hauteur b)
;;
```

▶ Question 3

```
let etiquettes t =
  let rec aux l = function
    | Empty -> l
    | Node(a,k,b) -> aux (k::(aux l b)) a
  in
  aux [] t
;;
```

▶ Question 4

```
let rec est_dans x = function
  | Empty -> false
  | Node(a,k,b) -> k==x||(est_dans x a)||(est_dans x b)
;;
```

▶ Question 5

```
let rec insere x = function
  | Empty -> Node(Empty,x,Empty)
  | Node(a,k,b) ->
    if x<k
    then Node(insere x a,k,b)
    else Node(a,k,insere x b)
;;
```

▶ Question 6

```
let tri l = etiquettes (it_list (fun t x -> insere x t) Empty l) ;;
```

▶ Question 7

On remarque que la hauteur est (empiriquement) de $O(n\times log(n))$

▶ Question 8

```
let unit t1 t2 = it_list (fun t x -> insere x t) t1 (etiquettes t2) ;;
```

▶ Question 9

```
let rec extrait_min = function
  | Empty -> failwith "Wut?"
  | Node(Empty,k,b) -> (k,b)
  | Node(a,k,b) -> let m,t = extrait_min b in (m,Node(a,k,t))
;;
let rec supprime x = function
  | Empty -> failwith "Wut?"
  | Node(a,k,b) when k<>x ->
    if x < k
    then Node(supprime x a,k, b)
    else Node(a,k,supprime x b)
  | Node(a,k,Empty) -> a
  | Node(a,k,b) -> let m,t = extrait_min b in Node(a,m,t)
;;
```

▶ Question 10

```
let rotg = function
  | Node(a,x,Node(b,p,c)) -> Node(Node(a,x,b),p,c)
  | _ -> failwith "rotg impossible"
let rotd = function
  | Node(Node(a,x,b),p,c) -> Node(a,x,Node(b,p,c))
  | _ -> failwith "rotd impossible"
;;
```

▶ Question 11

```
type 'a treap == ('a*int) tree;;
```

▶ Question 12

```
let rec est_dans_treap x = function
  | Empty -> false
  | Node(a,(k,p),b) ->
    if(x<k)
    then est_dans_treap x a
    else k = x || (est_dans_treap x b)
```

▶ Question 13

```
let respecte x = function
  | Empty -> true
  | Node(_,(_,k),_) -> k<=x
```

▶ Question 14

```
let tamise t = match t with
  | Empty -> Empty
  | Node(a,(k,p),b) ->
    if not respecte p a
    then rotd t
    else
      if not respecte p b
      then rotg t
      else t
```

▶ Question 15

```
let rec insere x = function
  | Empty -> Node(Empty,(x,random__int 100000000),Empty)
  | Node(a,(k,p),b) ->
    tamise (if x < k
            then Node(insere x a,(k,p),b)
            else Node(a,(k,p),insere x b))
```

▶ Question 16

```
let rec suppr x = function
  | Empty -> failwith "wat?"
  | Node(a,(k,p),b) when k<>x ->
    if k < x
    then Node(suppr x a,(k,p),b)
    else Node(a,(k,p),suppr x b)
  | Node(Empty,(k,p),b) -> b
  | Node(Node(aa,(ka,pa),ba),(k,p),b) ->
  Node(suppr ka (Node(aa,(ka,pa),ba)),(ka,pa),b)
```

▶ Question 17

```
let split x t = match tr_insere ((1000*1000*1000),x) t with
  | Vide -> (Vide,Vide)
  | Treap(f1,m,f2) -> (f1,f2)
;;
```