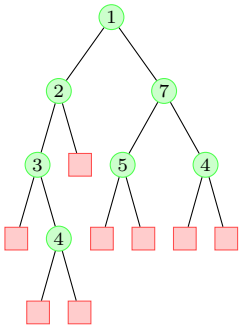
Algorithmique des structures de données arborescentes

Feuille d'exercices 3

1 Arbres Binaires : parcours

Exercice 3.1 Ecrire une fonction `btree_display` de type `val btree_display : int btree -> unit = <fun>` pour afficher en "mode texte" un arbre binaire.
Pour l'exemple de l'arbre binaire `btree_feuille_01` de la Fig. 3.1(a), l'affichage est illustré sur la Fig. 3.1(b).




```
let _ = btree_display btree_feuille_01;;
            -> Empty
        ->4
            -> Empty
    ->7
            -> Empty
        ->5
            -> Empty
->1
        -> Empty
    ->2
                -> Empty
            ->4
                -> Empty
        ->3
            -> Empty

- : unit = ()
```

FIG. 3.1 : (a) Arbre binaire (b) Affichage mode text

Exercice 3.2 On définit le type suivant permettant de représenter des squelettes d'arbres binaires complets, c'est-à-dire des arbres binaires complets ne contenant aucune donnée.

```
type stree =
  | Empty
  | Node of stree * stree
```

À tout squelette `t` d'arbre binaire complet (c'est-à-dire que chaque noeud de `t` a zéro ou exactement deux fils), on associe un mot binaire défini de la manière suivante :

- `encode(t)="1"`, si `t` est l'arbre vide,
- `encode(t)="0" ^ encode(t1) ^ encode(t2)`, si `t` a pour fils gauche `t1` et pour fils droit `t2`.

L'opérateur `^` désigne la concaténation de chaînes de caractères.

1. Écrire la fonction `encode` de type `val encode : stree -> string = <fun>` qui retournera une chaîne de caractères composées de 0 et de 1 codant le squelette `t` d'arbre binaire complet.
   Exemple :

```
let _ = encode(Empty);; (* - : string = "1" *)
let _ = encode(Node(Empty,Empty));; (* - : string = "011"  *)
let t = Node (Node (Node (Empty, Empty), Node (Empty, Empty)), Empty);;
let _ = encode(t);; (*- : string = "000110111" *)
```

2. Écrire la fonction `decode(wbin)` qui étant donné un mot binaire retourne le squelette d'arbre binaire complet codé par ce mot.
   On pourra s'aider d'une fonction auxiliaire récursive qui s'applique à un mot binaire `w` et qui retourne un couple composé de deux valeurs :

— un squelette d'arbre binaire complet dont le code est un mot `u` préfixe de `w`,
— le suffixe de `w` qui reste à décoder (c'est-à-dire le mot `v` tel que `w = u^v`.

```
let _ = decode("1");; (* -> - : stree = Empty *)
let _ = decode("011");; (* -> - : stree = Node (Empty, Empty) *)
let _ = decode("000110111");;
(* - : stree = Node (Node (Node (Empty, Empty), Node (Empty, Empty)), Empty);; *)
let _ = decode(encode(t)) = t ;; (* - : bool = true *)
```

On pourrait utiliser les fonctions `String.length` et `String.sub` du module `String`.

```
val length : string -> int
val sub : string -> int -> int -> string
```

`String.sub s start len` renvoie une sous-chaîne de la chaîne `s` qui commence avec le caractère en position `start` et qui est de longueur `len`.

2 Preuves de propriétés par récurrence

Exercice 3.3 Un arbre binaire t est dit *1-équilibré* si, pour ***tout*** nœud x de l'arbre t, la différence entre la hauteur du sous-arbre gauche de x et la hauteur du sous-arbre droit de x est comprise entre -1 et 1.

Pour un entier $h \geqslant 0$, on note $\mathsf{m}(h)$ le nombre ***minimal*** de nœuds dans un arbre 1-équilibré de hauteur h.

1. Calculer $\mathsf{m}(0)$, $\mathsf{m}(1)$, $\mathsf{m}(2)$ et $\mathsf{m}(3)$.
2. Trouver une relation de récurrence liant $\mathsf{m}(h+2)$, $\mathsf{m}(h+1)$ et $\mathsf{m}(h)$.
3. On définit la suite $(u_h)_{h\in\mathbb{N}}$ par $u_h = 1 + \mathsf{m}(h)$. Quelle relation de récurrence la suite u_h satisfait-elle ?
4. En déduire qu'il existe une constante $\alpha > 0$ telle que $\mathsf{m}(h) > \alpha(\sqrt{2})^h$.
5. (**Facultatif**) Montrer qu'il existe une constante $\beta > 0$ telle que $\mathsf{m}(h) > \beta \cdot \varphi^h$, où $\varphi = \dfrac{1+\sqrt{5}}{2} \approx 1\,.618$.
6. En déduire qu'il existe une constante $\gamma > 0$ telle que tout arbre 1-équilibré à n nœuds a une hauteur au maximum $\gamma \cdot \log_2(n)$.

Lorsqu'on ajoute à un arbre 1-équilibré la propriété d'être un arbre binaire de recherche (voir cours 3), on obtient les arbres AVL, pour lesquels cette dernière propriété conduit à des algorithmes très efficaces de recherche/insertion/suppression d'une valeur dans un ensemble. est importante pour obtenirLe terme AVL provient des initiales des chercheurs ayant introduit ces arbres, Georgii Adelson-Velsky et Evguenii Landis.

3 TD machine

Tester les fonctions `btree_display`, `encode` et `decode`.

Exercice 3.4 Dans cet exercice, on utilise le type `'a btree` suivant :

```
type 'a btree = Empty | Node of 'a * 'a btree * 'a btree
```

Si `p : int -> int -> bool` est un prédicat qui associe un booléen à deux entiers, on dit qu'un arbre `t` est `p`-équilibré si, pour tout nœud `x` de `t` , on a `p h_left h_right`, en notant `h_left` la hauteur du sous-arbre gauche de `x` et `h_right` la hauteur de son sous-arbre droit.

1. Écrire une fonction

```
val btree_is_balanced : ('a -> 'a -> bool) -> 'b btree -> ('b btree -> 'a) -> bool = <fun>
```

telle que `btree_is_balanced p t` retourne `true` si `t` est `p`-équilibré et `false` sinon.

2. Utiliser la fonction `btree_is_balanced` pour écrire une fonction `btree_is_perfect` qui teste si un arbre est parfait.

```
val btree_is_perfect : 'a btree -> bool = <fun>
```

3. En utilisant la fonction `btree_is_balanced`, écrire une fonction `btree_is_avl` qui teste si un arbre binaire est un arbre 1-équilibré.

```
val btree_is_avl : 'a btree -> bool = <fun>
```