Université de Bordeaux **Licence S&TS Mention Informatique, S4**
4TIN401U **2018/2019**

Algorithmique des structures de données arborescentes
TP noté du Mercredi 17 avril
Durée : 1h20.

Le barème est donné à titre **indicatif**.
Le sujet comporte 3 pages plus une annexe.

Consignes

1. Depuis `Moodle`, téléchargez et décompressez l'archive `NOM-PRENOM.zip` contenant le squelette du tp.
2. Renommez le répertoire `NOM-PRENOM` en remplaçant `NOM` et `PRENOM` par vos nom et prénom. Exemple : `TURING-ALAN`.
3. Placez-vous dans le répertoire ainsi renommé. Ce répertoire contient entre autres le fichier `tp.ml`.
4. Indiquez à nouveau vos nom et prénom et votre numéro de groupe dans le fichier `tp.ml`.

Les fichiers `exemples.ml` et `test.ml` sont des aides pour tester votre code. Le premier contient des expressions à tester. Le second permet de tester que les expressions retournent bien la valeur attendue.

5. Pensez à sauver régulièrement votre travail.
6. À la fin du tp,
   (a) Déposez le fichier `tp.ml` à l'endroit prévu sur Moodle.
   (b) Créez une archive `<NOM>-<PRENOM>.zip` à partir de votre répertoire.
   (c) Envoyez l'archive à votre chargé de TD :
   `irene.durand@u-bordeaux.fr` ou `simon.archipoff@u-bordeaux.fr`

Soit B, l'ensemble des mots écrits sur l'alphabet à 2 lettres $\{0, 1\}$. La *longueur* d'un mot est son nombre de lettres. Par exemple, 0010 est un mot de longueur 4.
En `OCaML`, nous utiliserons les entiers `0` et `1` pour représenter les lettres 0 et 1 et une liste de ces entiers pour représenter un mot. Ainsi, le *mot vide* (de longueur 0) est représenté par la liste vide `[]` et le mot 011001 par la liste `[0; 1; 1; 0; 0; 1]`.
Pour optimiser la recherche et économiser l'espace mémoire, on souhaite représenter un sous-ensemble de B (donc un sous-ensemble de mots binaires) de manière factorisée.
Pour cela on utilise un arbre binaire dont le type est le suivant :

```
type tree = Empty | Node of bool * tree * tree
```

— Le constructeur `Empty` représente l'ensemble vide de mots `[]`.

Si `TL` et `TR` représentent respectivement les ensembles de mots `L = [l1; ...; lp]` et `R = [r1; ...; rq]` alors

— `Node(false, TL, TR)` représente l'ensemble `[0::l1; ...; 0::lp; 1::r1; ...; 1::rq]`

— `Node(true, TL, TR)` représente l'ensemble `[[]; 0::l1; ...; 0::lp; 1::r1; ...; 1::rq]`

Remarques :

1. les mots se lisent dans l'arbre à partir de la racine ; un sous-arbre gauche correspond à un `0` et un sous-arbre droit correspond à un `1`.
2. Le booléen du constructeur `Node` permet d'indiquer si le noeud correspond à la fin d'un mot.
3. Le nombre de noeuds ayant le booléen à `true` est égal au cardinal de l'ensemble de mots représenté.

4. La longueur de la branche la plus longue est égale à la longueur du mot le plus long de l'ensemble de mots représenté.
5. `Node(true, Empty, Empty)` code l'ensemble contenant uniquement le mot vide.

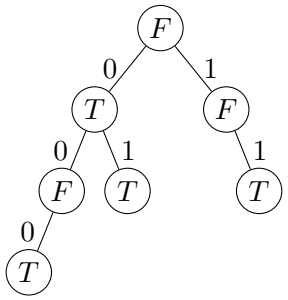

FIGURE 1 – `[[0]; [0; 0; 0]; [0; 1]; [1; 1]]`

Exemples :

— L'ensemble `[[0]; [0; 0; 0]; [0; 1]; [1; 1]]` est codé par

```
Node (false,
 Node (true, Node (false, Node (true, Empty, Empty), Empty),
  Node (true, Empty, Empty)),
 Node (false, Empty, Node (true, Empty, Empty)))
```

et illustré Figure 1.

— L'ensemble `[[]; [0; 0]; [1]; [1; 0; 1]]` est codé par

```
Node (true, Node (false, Node (true, Empty, Empty), Empty),
 Node (true, Node (false, Empty, Node (true, Empty, Empty)), Empty))
```

et illustré Figure 2.

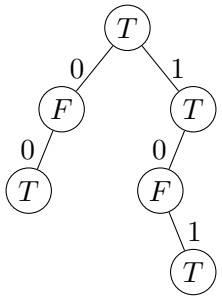

FIGURE 2 – `[[]; [0; 0]; [1]; [1; 0; 1]]`

Des exemples d'utilisation des fonctions demandées par la suite sont donnés en annexe.

Exercice 1 *Recherche (5 pts)*

1. Écrire la fonction `tree_mem word tree` de type `int list -> tree -> bool` qui retourne `true` si le mot `word` est présent dans l'arbre `tree` et `false` sinon.

Une feuille de l'arbre dont le booléen est à `false` est appelée une *feuille morte* car elle ne correspond à aucun mot. Pour obtenir un codage unique d'un ensemble de mots, on **impose** qu'il n'y ait pas de feuille morte dans l'arbre donc pas d'occurrence de `Node(false, Empty, Empty)`. Toute occurence de `Node(false, Empty, Empty)` doit être remplacée par `Empty`.

Exercice 2 *Ajout du mot vide (2 pts)*

2. Écrire la fonction `set_is_end tree end_word` de type `tree -> bool -> tree` qui retourne l'arbre `tree` auquel on a ajouté le mot vide [1].

Exercice 3 *Insertion (7 pts)*

3. Écrire la fonction `tree_insert word tree` de type `int list -> tree -> tree` qui retourne l'arbre `tree` auquel on a *ajouté* le mot `word`.
4. Écrire une fonction `tree_of_words` de type `int list list -> tree` qui retourne l'arbre contenant exactement tous les mots de la liste `words`.

Exercice 4 *Décodage (6 pts)*

5. Écrire la fonction `distribute b words` de type `'a -> 'a list list -> 'a list list` qui étant donné une liste de mots `[w1; ...; wn]` retourne `[b::w1; ...; b::wn]`
6. Écrire la fonction `words_of tree word` qui retourne sous forme de liste l'ensemble des mots codés pas `tree`.

Nous nous intéressons maintenant à la suppression. Pour supprimer un mot qui se termine dans un noeud qui n'est pas à une feuille, il suffit de mettre le booléen du noeud à faux. Pour supprimer un mot non vide qui aboutit à une feuille, il conviendra d'éliminer la feuille morte obtenue. d'éliminer.

Exercice 5 *Complexité (Bonus)*

7. Dans le cas où les mots que l'on insère sont les bits d'entiers sur 32 bits, quelle est la complexité de l'insertion ?

Exercice 6 *Suppression (Bonus)*

8. Écrire la fonction `tree_remove word tree` de type `int list -> tree -> tree` qui retourne l'arbre `tree` duquel on a *supprimé* le mot `word`.

1. c'est-à dire mis le booléen de la racine à `true`.

Annexe

```
utop[1]> let words1 = [[0]; [0; 0; 0]; [0; 1]; [1; 1]];;
val words1 : int list list = [[0]; [0; 0; 0]; [0; 1]; [1; 1]]
utop[2]> let words2 = [[]; [0; 0]; [1; 0; 1]];;
val words2 : int list list = [[]; [0; 0]; [1; 0; 1]]
utop[3]> let tree1 =
  Node (false,
        Node (true, Node (false, Node (true, Empty, Empty), Empty),
              Node (true, Empty, Empty)),
        Node (false, Empty, Node (true, Empty, Empty)));;
val tree1 : tree =
  Node (false,
   Node (true, Node (false, Node (true, Empty, Empty), Empty),
    Node (true, Empty, Empty)),
   Node (false, Empty, Node (true, Empty, Empty)))
utop[4]> let tree2 =
  Node (true,
        Node (false, Node (true, Empty, Empty), Empty),
        Node (false, Node (false, Empty, Node (true, Empty, Empty)), Empty));;
val tree2 : tree =
  Node (true, Node (false, Node (true, Empty, Empty), Empty),
   Node (false, Node (false, Empty, Node (true, Empty, Empty)), Empty))
utop[5]> let tree1_bis = set_is_end tree1;;
val tree1_bis : tree =
  Node (true,
   Node (true, Node (false, Node (true, Empty, Empty), Empty),
    Node (true, Empty, Empty)),
   Node (false, Empty, Node (true, Empty, Empty)))
utop[6]> let _ = tree_mem  [] tree1;;
- : bool = false
utop[7]> let _ = tree_mem  [] tree2;;
- : bool = true
utop[8]> let _ = tree_mem  [0; 0; 0] tree1;;
- : bool = true
utop[9]> let _ = tree_mem  [0; 0; 0] tree2;;
- : bool = false
utop[10]> let _ = tree_mem [0; 0] tree1;;
- : bool = false
utop[11]> let _ = tree_mem [0; 0] tree2;;
- : bool = true
utop[12]> let _ = tree_mem [1; 0] tree1;;
- : bool = false
utop[13]> let _ = tree_mem [1; 0] tree2;;
- : bool = false
utop[14]> let _ = distribute 0 words1;;
- : int list list = [[0; 0]; [0; 0; 0; 0]; [0; 0; 1]; [0; 1; 1]]
utop[15]> let _ = distribute 1 words1;;
- : int list list = [[1; 0]; [1; 0; 0; 0]; [1; 0; 1]; [1; 1; 1]]
utop[16]> let t11  = tree_insert [0] Empty;;
val t11 : tree = Node (false, Node (true, Empty, Empty), Empty)
```

```
utop[17]> let t12 = tree_insert [0; 0; 0] t11;;
val t12 : tree =
  Node (false,
   Node (true, Node (false, Node (true, Empty, Empty), Empty), Empty), Empty)
utop[18]> let t13 = tree_insert [0; 1] t12;;
val t13 : tree =
  Node (false,
   Node (true, Node (false, Node (true, Empty, Empty), Empty),
    Node (true, Empty, Empty)),
   Empty)
utop[19]> let t14 = tree_insert [1; 1] t13;;
val t14 : tree =
  Node (false,
   Node (true, Node (false, Node (true, Empty, Empty), Empty),
    Node (true, Empty, Empty)),
   Node (false, Empty, Node (true, Empty, Empty)))
utop[20]> let _ = tree1 = t14;;
- : bool = true
utop[21]> let t21  = tree_insert [] Empty;;
val t21 : tree = Node (true, Empty, Empty)
utop[22]> let t22 = tree_insert [0; 0] t21;;
val t22 : tree =
  Node (true, Node (false, Node (true, Empty, Empty), Empty), Empty)
utop[23]> let t23 = tree_insert [1; 0; 1] t22;;
val t23 : tree =
  Node (true, Node (false, Node (true, Empty, Empty), Empty),
   Node (false, Node (false, Empty, Node (true, Empty, Empty)), Empty))
utop[24]> let _ = tree2 = t23;;
- : bool = true
utop[25]> let t1 = tree_of_words words1;;
val t1 : tree =
  Node (false,
   Node (true, Node (false, Node (true, Empty, Empty), Empty),
    Node (true, Empty, Empty)),
   Node (false, Empty, Node (true, Empty, Empty)))
utop[26]> let _ = t1 = tree1;;
- : bool = true
utop[27]> let t2 = tree_of_words words2;;
val t2 : tree =
  Node (true, Node (false, Node (true, Empty, Empty), Empty),
   Node (false, Node (false, Empty, Node (true, Empty, Empty)), Empty))
utop[28]> let _ = t2 = tree2;;
- : bool = true
utop[29]> let _ = words_of t1;;
- : int list list = [[0]; [0; 0; 0]; [0; 1]; [1; 1]]
utop[30]> let _ = words_of t2;;
- : int list list = [[]; [0; 0]; [1; 0; 1]]
```