université de BORDEAUX	**Année 2017/2018** — L2 Info/MI, UE 4TIN402U Algorithmique des structures de données arborescentes Vendredi 4 mai 2018, 9h–10h30 — Durée : 1h30 Documents : **non autorisés** Le barème est donné à titre **indicatif**	**Collège Sciences et Technologies**

Exercice 1 *Arbres planaires et chemins de Dyck (4 points)*
On représente un mot de Dyck par une liste de directions :

```
type direction = Up | Down
type dyckpath = direction list
```

On rappelle que la bijection entre squelettes d'arbres planaires à $n+1$ nœuds et mots de Dyck de taille $2n$ est obtenue en parcourant l'arbre en profondeur, de la racine à la racine, et en produisant `Up` lorsque l'on descend le long d'une arête et `Down` lorsqu'on monte le long d'une arête.
Une illustration est donnée sur la Figure 1 (a).

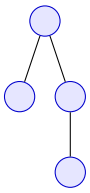
1. Écrire le mot de Dyck correspondant à l'arbre planaire de la Figure 1 (b).

On considère le type suivant pour représenter les arbres planaires non vides :

```
type planar_tree = Node of planar_tree list
```

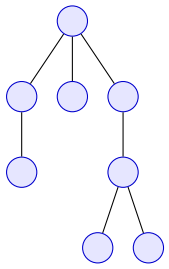
Par exemple, l'arbre constitué de 4 nœuds, une racine ayant 3 fils, est `Node[Node[];Node[];Node[]]`.

2. Écrire une fonction `planar_to_dyck : planar_tree -> dyckpath` produisant le mot de Dyck correspondant à un arbre planaire passé en argument.



Arbre planaire correspondant au mot de Dyck `[Up;Down;Up;Up;Down;Down]`

(a)



(b)

FIGURE 1 – Arbres planaires et mots de Dyck

Exercice 2 *Arbres rouges et noirs (4 points)*
Quel est l'arbre rouge et noir produit par l'algorithme d'insertion vu en cours lorsqu'on insère les entiers 7, 6, 5, 3, 4, 2, 1 dans cet ordre, à partir de l'arbre rouge et noir ne contenant aucune valeur ? Détailler chaque étape d'insertion (il n'est pas utile de représenter les feuilles, puisqu'elles ne contiennent pas de valeur).

Exercice 3 *Algorithme de Huffman (3 points)*
On considère le texte suivant :

OVERNERVOUSNESSES

Appliquer l'algorithme de Huffman vu en cours sur ce texte pour

1. Créer l'arbre de Huffman permettant d'associer un code à chaque caractère.
2. Produire la suite de bits correspondant au texte et la liste de couples (caractere, code).

Exercice 4 *Largeur d'un arbre (6 points)*
Dans cet exercice, on utilise le type suivant pour représenter des arbres binaires :

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

Par ailleurs, on suppose fourni un type `'a queue` permettant de représenter des **files** dont les éléments sont de type `'a`. On suppose les fonctions suivantes déjà écrites :

```
val empty_queue : 'a queue
val enqueue : 'a -> 'a queue -> 'a queue
val dequeue :'a queue -> ('a * 'a queue)
val queue_size : 'a queue -> int
```

La fonction `empty_queue` renvoie une file vide. L'appel `enqueue` x q renvoie la file obtenue lorsqu'on enfile l'élément x dans la file q, et la fonction `dequeue` renvoie un couple (x, q) où x est le premier élément de la file passée en argument et q est la file obtenue en défilant x de son argument (cette fonction produit une erreur si la file est vide). Enfin, la fonction `queue_size` renvoie le nombre d'éléments de son argument.

1. On considère le code suivant :

```
let rec next_level q acc_nodes acc_trees =
  if queue_size q = 0  then acc_nodes, acc_trees
  else let (t,q') = dequeue q in
    match t with
      Empty -> next_level q' acc_nodes acc_trees
    | Bin(x, left, right) ->
             next_level q' (x::acc_nodes) (enqueue right (enqueue left acc_trees))
```

   Simuler l'exécution du code suivant (décrire l'arbre `t` et les valeurs successives de `acc_nodes` et `acc_trees`).

```
let t = Bin(1, Bin(2, leaf 3, Bin(4, Bin(5, leaf 6, leaf 7), Empty)),
               Bin(8, Empty, leaf 9))
let acc_nodes,acc_trees = [],(enqueue t empty_queue)
let acc_nodes, acc_trees = next_level acc_trees acc_nodes empty_queue
let acc_nodes, acc_trees = next_level acc_trees acc_nodes empty_queue
```

2. Utiliser la fonction précédente pour écrire une fonction prenant en argument un arbre `t` et produisant la liste de ses nœuds dans un parcours en profondeur.
3. On définit la *largeur* d'un arbre non vide comme le nombre maximal de nœuds se trouvant à une même profondeur. Par exemple, la largeur de l'arbre de la question 1 vaut 3, car il y a 3 nœuds à profondeur 2, plus que dans toutes les autres profondeurs.

   Modifier l'algorithme de la question 2 pour qu'il calcule la largeur de l'arbre.

Exercice 5 *Preuve de propriété (3 points)*
Soit t un arbre binaire **plein** à n nœuds. On suppose que $n \geqslant 2$. Montrer par récurrence sur la hauteur de t que t possède deux feuilles qui ont la même profondeur.