TD #11 — Arbres binaires

Exercice 1. *Parcours en profondeur en C*

On considère le type :

```
struct btree {
    struct btree *left; struct btree *right; int data;
};
```

1. Écrivez une fonction C de signature `void dfsi(struct btree *t)` qui ne fait rien si son argument est un pointeur nul, et sinon effectue un parcours en profondeur infixe de l'arbre représenté par l'objet référencé par son argument afin d'afficher les contenus `.data` de ses nœuds.
2. Donnez un extrait de code construisant un arbre pour lequel la fonction précédente affiche `1 2 3`, et effectue cet appel.

Exercice 2. *Relation feuille-nœud*

Montrez la *relation feuille-nœud* pour un arbre binaire, c'est à dire que le nombre de feuilles F et de nœuds internes N d'un arbre binaire satisfont l'inégalité $F \leq N + 1$, avec égalité ssi. l'arbre est strict (c'est à dire est tel que chacun de ses nœuds internes a exactement deux enfants).

Exercice 3. *Numérotation par mots binaire*

Dans tout cet exercice, on utilisera le `type tree = E | N of tree * tree` pour représenter les arbres.

On rappelle une numérotation par mots binaires des nœuds d'un arbre binaire :

— la racine (non vide) est numérotée par le mot vide ε (élément neutre pour la concaténation « $\cdot$ ») ;
— l'enfant gauche (resp. droit) non vide d'un nœud de numéro ν est numéroté $0 \cdot \nu$ (resp. $1 \cdot \nu$).

1. Soit `t1`, `t2` deux arbres binaires et `t = N(t1, t2)`, montrez que pour tout nœud non vide `t1'` de `t1` (resp. `t2'` de `t2`), soit ν_1 (resp. ν_2) son numéro dans `t1` pour la numérotation ci-dessus, alors son numéro dans `t` est $\nu_1 \cdot 0$ (resp. $\nu_2 \cdot 1$).
2. Montrez que cette numérotation est injective, c'est à dire que tous les nœuds (non vides) d'un arbre ont un numéro distinct pour cette numérotation.
3. Proposez (informellement) un algorithme récursif efficace qui étant donnée (la racine d') un arbre binaire `t` et un numéro ν accède au nœud de numéro correspondant dans `t` (s'il existe).

Exercice 4. *Parcours en largeur*

Un *parcours en largeur* d'un arbre binaire est un algorithme qui visite tous les nœuds de l'arbre par niveau de profondeur croissant, c'est à dire qu'il visitera d'abord la racine (de profondeur 0), puis tous les enfants de la racine (de profondeur 1), puis tous les petits-enfants (de profondeur 2), etc.

On souhaite montrer dans cet exercice qu'un tel parcours peut être réalisé efficacement de la façon suivante : on initialise une *file* avec la racine de l'arbre, puis tant que la file est non vide on défile un élément, le visite, puis enfile ses (éventuels) enfants.

1. Implémentez l'algorithme esquissé ci-dessus en OCaml par une fonction :
   `bfs : ('a -> unit) -> 'a tree -> unit`
   pour le `type 'a tree = E | N of 'a * 'a tree * 'a tree`. Cette implémentation *devra* être écrite en style impératif et utiliser l'implémentation de file (impérative) fournie par le module `Queue`, notamment les fonctions :
   - `Queue.create : unit -> 'a queue`
   - `Queue.is_empty : 'a queue -> bool`
   - `Queue.push : 'a -> 'a queue -> unit`
   - `Queue.pop : 'a queue -> 'a`
2. Montrez que si la file contient uniquement des arbres vides `E`, la boucle `while` n'ajoute aucun nouvel élément à la file, et termine.
3. Montrez que la boucle `while` de votre fonction satisfait l'invariant de boucle suivant : la file est ou bien vide, ou bien contient uniquement des arbres vides, ou bien contient (en plus d'éventuels arbres vides) des nœuds de profondeurs toutes égales, ou bien des nœuds de profondeur d et $d+1$, où tous les nœuds de profondeur d ont été insérés avant ceux de profondeur $d+1$.
4. Montrez que lors d'une itération de la boucle `while`, l'état successeur d'une file qui (en ignorant les arbres vides) contient uniquement des nœuds de profondeur d est une file contenant ou bien uniquement des arbres vide, ou bien contenant des nœuds de profondeur d et $d+1$, ou bien contenant des nœuds de profondeur $d+1$.
5. Montrez que si `t` est non vide, le premier élément à être défilé est le nœud de profondeur zéro (la racine) de `t`.
6. Montrez que la suite des profondeurs des nœuds (on ignore donc les arbres vides) extraits de la file est croissante, et si non vide contient tous les entiers $\in [\![0, h]\!]$ pour un certain h.
7. Montrez que tout nœud est visité exactement une fois par ce parcours.
8. Conclure.