Algorithmique des structures de données arborescentes

Feuille d'exercices 4

4.1 Arbres binaires de recherche (suite)

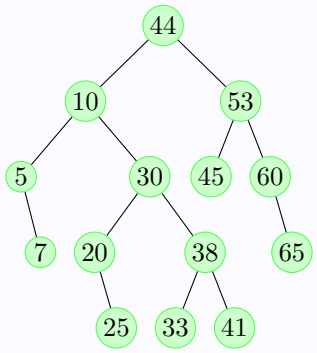
On utilisera le type déjà vu précédemment pour coder les arbres binaires en OCaml :

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

Exercice 1 : Max

1. Proposer un algorithme pour trouver l'élément maximal dans un ABR.
2. Écrire une fonction `bst_max` de type `int btree -> int` qui renvoie la cle maximale de l'arbre, si l'ABR est non vide et `failwith "Empty tree "` si l'ABR est vide.

Exercice 2 : Suppression dans les arbres binaires de recherche



1. Proposer une réorganisation de l'ABR ci-dessus, après supression de la cle 30, de sorte que le résultat obtenu reste un ABR.
2. Proposer un algorithme pour supprimer une cle dans un ABR. Lorsqu'il faut supprimer un nœud, distinguer également plusieurs cas selon que le nœud a zéro, un ou deux fils vides.
3. Écrire une fonction `bst_pop_max` de type `'a btree -> 'a btree * 'a` qui prend en argument un ABR `t` et renvoie un couple `(t', c)` où `c` est la cle maximale de `t` et `t'` est un ABR obtenu en suprimant la cle `c` de `t`.
4. Écrire une fonction `bst_remove` de type `'a btree -> 'a -> 'a btree` qui renvoie l'ABR original si la cle donnée en second argument n'y est pas présente, et l'ABR obtenu en supprimant cette cle sinon.

Exercice 3 : Caractérisations des arbres binaires de recherche

1. Proposer un algorithme pour tester qu'un arbre binaire est un ABR.
2. Comment modifier l'algorithme pour obtenir une complexité linéaire en fonction de la taille de l'arbre ?
3. Écrire une fonction `is_bst` de type `'a btree -> bool` qui teste si un arbre est un ABR.