Structures de données et algorithmes

Feuille de TD n°5

Arbres binaires de recherche (ABR)

Exercice 1. Tri par ABR

a) Dans le parcours en profondeur d'un ABR selon l'ordre infixe, à quoi correspond l'ordre de visite des nœuds ?

b) En utilisant la réponse à la question précédente, écrivez la définition d'une fonction `print_sorted_BST` qui reçoit deux entrées : un tableau de valeurs (de type `item`) et sa taille. Elle affiche les valeurs du tableau dans l'ordre croissant. Cette fonction construit un ABR à partir du tableau puis exécute un parcours en profondeur selon l'ordre infixe de l'ABR ainsi construit.

c) Évaluez le coût de l'exécution de la fonction écrite à la question précédente en fonction de la taille du tableau à trier. Comparez ce coût avec celui du tri par tas du TP noté du vendredi 24 novembre.

Exercice 2. Insertion itérative dans un ABR

On rappelle que la fonction `insert_BST` définie ci-dessous insère un nœud d'étiquette v comme une nouvelle feuille dans l'ABR d'adresse h et renvoie l'adresse de l'ABR mis à jour.

```
1  link insert_BST(link h, item v) {
2    if (h == NULL) return cons_binary_tree(v, NULL, NULL);
3    if less(v, get_binary_tree_root(h)) {
4      h->left = insert_BST(h->left, v);
5    }
6    else {
7      h->right = insert_BST(h->right, v);
8    }
9    return h;
10 }
```

a) Écrivez la définition d'un fonction itérative `insert_BST_it` qui dérécursive `insert_BST` **sans utiliser de pile**.

On rappelle ci-après la définition des fonctions qui implémentent les rotations gauche et droite d'un ABR et celle de la fonction `insert_root_BST` qui insère un nœud d'étiquette v à la racine de l'ABR d'adresse h et renvoie l'adresse de l'ABR mis à jour.

```
1  link rotate_left(link h) {
2    link x;
3    if (NULL == h) return NULL;
4    x = h->right;
5    h->right = x->left;
6    x->left = h;
7    return x;
```

```
link rotate_right(link h) {
  link x;
  if (NULL == h) return NULL;
  x = h->left;
  h->left = x->right;
  x->right = h;
  return x;
}
```

```
link insert_BST_root(link h, item v) {
  if (h == NULL) return cons_binary_tree(v, NULL, NULL);
  if (less(v, get_binary_tree_root(h))) {
    h->left = insert_BST_root(h->left, v);
    h = rotate_right(h);
  }
  else {
    h->right = insert_BST_root(h->right, v);
    h = rotate_left(h);
  }
  return h;
}
```

b) Écrivez la définition d'un fonction itérative `insert_root_BST_it` qui dérécursive `insert_root_BST` **en utilisant une pile**.

Exercice 3. Partitionnement et suppression d'un nœud dans un ABR

La fonction ci-dessous « partitionne » l'ABR d'adresse h en faisant remonter à la racine le nœud portant l'étiquette de rang k : dans l'ABR dont l'adresse est renvoyée par la fonction, le sous-arbre gauche de la racine a exactement k nœuds.

```
link partition_BST(link h, int k) {
  int t = size_binary_tree(h->left);
  if (is_empty_binary_tree(h)) return h;
  if (t > k) {
    h->left = partition_BST(h->left, k);
    h = rotate_right(h);
  }
  if (t < k) {
    h->right = partition_BST(h->right, k-t-1);
    h = rotate_left(h);
  }
  return h;
}
```

Écrivez la définition d'une fonction `delete_node_BST` qui reçoit deux entrées : l'adresse h d'un ABR et une valeur v. Elle supprime un nœud d'étiquette v et renvoie l'adresse de l'ABR mis à jour. Cette fonction appelle `partition_BST`.