

TP Algo2 – session 5 – Arbres binaires de recherche

23 mai 2025

Objectif d'apprentissage : implémenter la structure d'arbre binaire de recherche et les opérations de recherche et de modifications basiques (sur 2 séances de TP). Il faudra utiliser le code suivant pour les définitions et implémenter chaque méthode :

```
#ifndef BST_H
#define BST_H

#include <stddef.h>
#include <stdbool.h>

typedef struct sbst_node
{
    int key;
    struct sbst_node *left;
    struct sbst_node *right;
    struct sbst_node *parent;
} bst_node;

typedef struct
{
    bst_node *root;
} bst;

/*
 * Create an empty bst
 */
void bst_create (bst * self);

/*
 * Destroy a bst
 */
void bst_destroy (bst * self);

/*
 * Get the size of the tree
 */
size_t bst_size (const bst * self);

/*
 * Get the height of the tree
 */
size_t bst_height (const bst * self);

/*
 * Walk in the tree in inorder and print each value
 */
```

```

    */
void inorder_bst_walk (const bst * self);

/*
 * Find the node containing a value in the sub-tree rooted in self
 */
bst_node *bst_search (bst_node * self, int value);

/*
 * Find the node containing the minimum value in the sub-tree rooted
 * in self
 */
bst_node *bst_minimum (bst_node * self);

/*
 * Find the node containing the maximum value in the sub-tree rooted
 * in self
 */
bst_node *bst_maximum (bst_node * self);

/*
 * Find the predecessor containing of the given node
 */
bst_node *bst_predecessor (bst_node * self);

/*
 * Find the successor containing of the given node
 */
bst_node *bst_successor (bst_node * self);

/*
 * Insert a value in the tree
 */
void bst_insert (bst * self, int value);

/*
 * Remove a node from the tree
 */
void bst_delete (bst * self, bst_node * node);

#endif // BST_H

```

L'objectif de cette première séance sur ce sujet est d'implémenter et tester les méthodes jusqu'à `bst_maximum` compris. Comme précédemment, lorsqu'une fonction ne peut pas terminer son exécution (valeur absente ou argument à `NULL`), la fonction renverra une valeur par défaut : 0 si le type de retour est `int`, `NULL` si c'est un pointeur, `false` si c'est un booléen. On supposera cependant que si les pointeurs vers des structures `bst` sont définis (non nuls), alors les structures représentent des arbres binaires de recherche valides (tous les nœuds sont bien présents et sans redondance) et qu'il n'y a pas besoin de le vérifier dans `bst.c` (mais il faut s'assurer que les fonctions manipulent correctement ces structures dans `bst_tests.c`).

L'évaluation (facultative et à titre indicatif) consistera à soumettre deux fichiers : les implémentations (`bst.c`) et les tests unitaires (`bst_tests.c`). Si une fonction auxiliaire paraît utile pour les tests, c'est dans ce fichier de test qu'il faudra la rajouter.

L'évaluation de `bst.c` se fait sur la base du nombre d'avertissements générés par `gcc`, par `valgrind`, de la validité du code en utilisant les tests unitaires enseignants et sur le respect des conventions de codage avec `indent`.

Les deux fichiers doivent être soumis **tels quels** pour permettre le traitement et l'évaluation automatique (non compressés, non archivés, non renommés).