Exercises — Binary Tree: Dynamic Implementation

version **#7be580532266ed398481e31366afcc24b1950c2a**

Assistants C/Unix 2023 <assistants@tickets.assistants.epita.fr>

Copyright

This document is for internal use at EPITA (website) only.

Contents

*https://intra.assistants.epita.fr

File Tree

```
binary_tree_dynamic/
├── *  (to submit)
├── Makefile  (to submit)
└── binary_tree.h
```

Makefile

- library: Produces the libbinary_tree.a library
- clean: Deletes everything produced by make

Authorized functions : You are only allowed to use the following functions

- printf(3)

Authorized headers : You are only allowed to use the functions defined in the following headers

- err.h
- errno.h
- assert.h
- stddef.h
- limits.h

Compilation : Your code must compile with the following flags

- -std=c99 -pedantic -Werror -Wall -Wextra -Wvla

Main function : None

1 Goal

In this exercise you will implement some useful functions to work on binary trees. Your binary tree will have its nodes labeled with `int` values, and is implemented using the following structure:

```c
struct binary_tree
{
    int data;
    struct binary_tree *left;
    struct binary_tree *right;
};
```

The `NULL` pointer represents an empty tree.

2 Size

```c
int size(const struct binary_tree *tree);
```

This function returns the size (number of nodes) of `tree`.

3 Height

```c
int height(const struct binary_tree *tree);
```

This function returns the height of `tree`.

4 Depth-first traversal

```c
void dfs_print_prefix(const struct binary_tree *tree);
void dfs_print_infix(const struct binary_tree *tree);
void dfs_print_postfix(const struct binary_tree *tree);
```

Those functions, also known as Depth-first traversal, displays labels of the nodes of `tree` in the specified order when doing a left-to-right traversal. The functions print the values separated with a space and a trailing white space and *NO* return line.

For example, the output could look like this:

```
42sh$ ./example | cat -e
1 2 3 42sh$
```

If the tree is empty, you have to print an empty string.

```
42sh$ ./example | cat -e
42sh$
```

5 Is perfect

```c
int is_perfect(const struct binary_tree *tree);
```

This function returns `1` if the tree `tree` is perfect, `0` otherwise.

6 Is complete

```
int is_complete(const struct binary_tree *tree);
```

This function returns `1` if the tree `tree` is complete, `0` otherwise.

7 Is degenerate

```
int is_degenerate(const struct binary_tree *tree);
```

This function returns `1` if the tree `tree` is a degenerate tree, `0` otherwise.

A degenerate tree is a tree where each node has at most one child.

8 Is full

```
int is_full(const struct binary_tree *tree);
```

This function returns `1` if the tree `tree` is full, `0` otherwise.

A full binary tree is a tree where each node is either a leaf or has two child.

9 Is BST

```
int is_bst(const struct binary_tree *tree);
```

This function returns `1` if all nodes in the binary tree `tree` are sorted according to a total order (the tree is a binary search tree), `0` otherwise.

The way is lit. The path is clear. We require only the strength to follow it.