UCSD 167113 Data Structure And Algorithms in C/C++
HW6
Cheng FEI

Source Code:

```
//
//  main.c
//  tree-searcher
//
//  Created by Cheng FEI on 2022/11/5.
//

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "bitree.h"

#define max(a, b) ((a) > (b) ? (a) : (b))

// Prototypes of Tree Initialization Functions.
int init_tree1(BiTree *tree, int data[]);
int init_tree2(BiTree *tree, int data[]);

// Prototypes of Major Tree Functions.
int count_leaves(BiTree *tree);
int count_non_leaves(BiTree *tree);
int count_node_leaves(BiTreeNode *node);
int get_height(BiTree *tree);
int get_node_height(BiTreeNode *node);
void print_pre_order(BiTree *tree, void (*print)(const void *data));
void print_node_pre_order(const void *node);
void print_in_order(BiTree *tree, void (*print)(const void *data));
void print_node_in_order(const void *node);
void print_post_order(BiTree *tree, void (*print)(const void *data));
void print_node_post_order(const void *node);
void remove_leaves(BiTree *tree);
void remove_node_leaves(BiTreeNode *node);

int main(int argc, const char * argv[]) {
    int init_stat;

    // First test case.
    // Initialize the tree.
    BiTree *tree1;
    if ((tree1 = (BiTree *) malloc(sizeof(BiTree))) == NULL) {
        fprintf(stderr, "Out of memory!");
        exit(EXIT_FAILURE);
    }
    int data1[] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
    init_stat = init_tree1(tree1, data1);
    if (init_stat == 0) {
        printf("Tree One Statistics:\n");
        printf("------------------------------------\n");
        printf("Total number of leaves:            %d\n", count_leaves(tree1));
        printf("Expected:                          3\n");
        printf("Total number of non-leaf nodes:    %d\n", count_non_leaves(tree1));
        printf("Expected:                          6\n");
        printf("Height of the tree:                %d\n", get_height(tree1));
        printf("Expected:                          5\n");
        printf("Display in pre-order:\n");
        print_pre_order(tree1, print_node_pre_order);
        printf("Expected:\n");
        printf("1 2 4 7 3 5 6 8 9\n");
        printf("Display in in-prder:\n");
        print_in_order(tree1, (void *) print_node_in_order);
        printf("Expected:\n");
        printf("7 4 2 1 5 3 6 8 9\n");
        printf("Display in post-order:\n");
        print_post_order(tree1, (void *) print_node_post_order);
```

```
        printf("Expected:\n");
        printf("7 4 2 5 9 8 6 3 1\n");
        printf("-----------------------------------\n");
        printf("Removing leaves...\n");
        remove_leaves(tree1);
        printf("-----------------------------------\n");
        printf("Remaining nodes:\n");
        print_pre_order(tree1, print_node_pre_order);
        printf("Expected:\n");
        printf("1 2 4 3 6 8\n");
        printf("\n\n");
    }
    else {
        fprintf(stderr, "Tree One initialization failed!");
    }
    // Release the memory.
    bitree_destroy(tree1);

    // Second test case.
    // Initialize the tree.
    BiTree *tree2;
    if ((tree2 = (BiTree *) malloc(sizeof(BiTree))) == NULL) {
        fprintf(stderr, "Out of memory!");
        exit(EXIT_FAILURE);
    }
    int data2[] = {6, 4, 8, 2, 5, 7, 9, 1, 3};
    init_stat = init_tree2(tree2, data2);
    if (init_stat == 0) {
        printf("Tree Two Statistics:\n");
        printf("-----------------------------------\n");
        printf("Total number of leaves:            %d\n", count_leaves(tree2));
        printf("Expected:                          5\n");
        printf("Total number of non-leaf nodes:    %d\n", count_non_leaves(tree2));
        printf("Expected:                          4\n");
        printf("Height of the tree:                %d\n", get_height(tree2));
        printf("Expected:                          4\n");
        printf("Display in pre-order:\n");
        print_pre_order(tree2, print_node_pre_order);
        printf("Expected:\n");
        printf("6 4 2 1 3 5 8 7 9\n");
        printf("Display in in-prder:\n");
        print_in_order(tree2, (void *) print_node_in_order);
        printf("Expected:\n");
        printf("1 2 3 4 5 6 7 8 9\n");
        printf("Display in post-order:\n");
        print_post_order(tree2, (void *) print_node_post_order);
        printf("Expected:\n");
        printf("1 3 2 5 4 7 9 8 6\n");
        printf("-----------------------------------\n");
        printf("Removing leaves...\n");
        remove_leaves(tree2);
        printf("-----------------------------------\n");
        printf("Remaining nodes:\n");
        print_pre_order(tree2, print_node_pre_order);
        printf("Expected:\n");
        printf("6 4 2 8\n");
        printf("\n\n");
    }
    else {
        fprintf(stderr, "Tree One initialization failed!");
    }
    // Release the memory.
    bitree_destroy(tree2);

    return 0;
}

// Initialize the first tree.
// @Params: tree -- pointer to the first tree.
// @Params: data -- integer array with values to insert.
// @Returns: 0 means a successful initialization.
//           1 means a failed initialization.
```

```
// @Requires: memory is allocated for tree.
int init_tree1(BiTree *tree, int data[]) {
    bitree_init(tree, NULL);
    // Insert the root node.
    if (bitree_ins_left(tree, NULL, data) != 0) return -1;
    // Insert nodes in the 1st level.
    if (bitree_ins_left(tree, tree->root, data+1) != 0) return -1;
    if (bitree_ins_right(tree, tree->root, data+2) != 0) return -1;
    // Insert nodes in the 2nd level.
    if (bitree_ins_left(tree, tree->root->left, data+3) != 0) return -1;
    if (bitree_ins_left(tree, tree->root->right, data+4) != 0) return -1;
    if (bitree_ins_right(tree, tree->root->right, data+5) != 0) return -1;
    // Insert nodes in the 3rd level.
    if (bitree_ins_left(tree, tree->root->left->left, data+6) != 0) return -1;
    if (bitree_ins_right(tree, tree->root->right->right, data+7) != 0) return -1;
    // Insert nodes in the 4th level.
    if (bitree_ins_right(tree, tree->root->right->right->right, data+8) != 0) return -1;

    return 0;
}

// Initialize the second tree.
// @Params: tree -- pointer to the second tree.
// @Params: data -- integer array with values to insert.
// @Returns: 0 means a successful initialization.
//           1 means a failed initialization.
// @Requires: memory is allocated for tree.
int init_tree2(BiTree *tree, int data[]) {
    bitree_init(tree, NULL);
    // Insert the root node.
    if (bitree_ins_left(tree, NULL, data) != 0) return -1;
    // Insert nodes in the 1st level.
    if (bitree_ins_left(tree, tree->root, data+1) != 0) return -1;
    if (bitree_ins_right(tree, tree->root, data+2) != 0) return -1;
    // Insert nodes in the 2nd level.
    if (bitree_ins_left(tree, tree->root->left, data+3) != 0) return -1;
    if (bitree_ins_right(tree, tree->root->left, data+4) != 0) return -1;
    if (bitree_ins_left(tree, tree->root->right, data+5) != 0) return -1;
    if (bitree_ins_right(tree, tree->root->right, data+6) != 0) return -1;
    // Insert nodes in the 3rd level.
    if (bitree_ins_left(tree, tree->root->left->left, data+7) != 0) return -1;
    if (bitree_ins_right(tree, tree->root->left->left, data+8) != 0) return -1;

    return 0;
}

// Count total number of the leaf node in the tree.
// @Params: tree -- pointer to the tree.
// @Returns: number of leaves in the tree.
int count_leaves(BiTree *tree) {
    if (tree == NULL) return 0;
    return count_node_leaves(tree->root);
}

// Count total number of the non-leaf node in the tree.
// @Params: tree -- pointer to the tree.
// @Returns: number of non-leaf nodes in the tree.
int count_non_leaves(BiTree *tree) {
    if (tree == NULL) return 0;
    return tree->size - count_leaves(tree);
}

// Count total number of the leaf node that is the descendant of input node.
// @Params: node -- pointer to the tree node.
// @Returns: number of the leaf node that is the descendant of input node.
int count_node_leaves(BiTreeNode *node) {
    if (node == NULL) return 0;
    if (bitree_is_leaf(node)) return 1;
    return count_node_leaves(node->left) + count_node_leaves(node->right);
}

// Calculate the height of the tree.
```

```
// @Params: tree -- pointer to the tree.
// @Returns: height of the tree.
int get_height(BiTree *tree) {
    if (tree == NULL) return 0;
    return get_node_height(tree->root);
}

// Calculate the height of the node.
// @Params: node -- pointer to the node.
// @Returns: height of the node.
int get_node_height(BiTreeNode *node) {
    if (node == NULL) return 0;
    if (bitree_is_leaf(node)) return 1;
    return max(get_node_height(node->left), get_node_height(node->right)) + 1;
    return 0;
}

// Display the tree in the pre-order format.
// @Params: tree -- pointer to the tree.
// @Params: print -- pointer to the pre-defined print function.
void print_pre_order(BiTree *tree, void (*print)(const void *data)) {
    if (tree == NULL) return;
    print(tree->root);
    printf("\n");
}

// Display nodes in the pre-order format.
// This is the helper function for @function{print_pre_order}.
// @Params: node -- pointer to the parent node.
void print_node_pre_order(const void *node) {
    BiTreeNode *treeNode = (BiTreeNode *) node;
    if (treeNode == NULL) return;
    printf("%d ", *((int *)treeNode->data));
    print_node_pre_order(treeNode->left);
    print_node_pre_order(treeNode->right);
}

// Display the tree in the in-order format.
// @Params: tree -- pointer to the tree.
// @Params: print -- pointer to the pre-defined print function.
void print_in_order(BiTree *tree, void (*print)(const void *data)) {
    if (tree == NULL) return;
    print(tree->root);
    printf("\n");
}

// Display nodes in the in-order format.
// This is the helper function for @function{print_in_order}.
// @Params: node -- pointer to the parent node.
void print_node_in_order(const void *node) {
    BiTreeNode *treeNode = (BiTreeNode *) node;
    if (treeNode == NULL) return;
    print_node_in_order(treeNode->left);
    printf("%d ", *((int *)treeNode->data));
    print_node_in_order(treeNode->right);
}

// Display the tree in the post-order format.
// @Params: tree -- pointer to the tree.
// @Params: print -- pointer to the pre-defined print function.
void print_post_order(BiTree *tree, void (*print)(const void *data)) {
    if (tree == NULL) return;
    print(tree->root);
    printf("\n");
}

// Display nodes in the post-order format.
// This is the helper function for @function{print_post_order}.
// @Params: node -- pointer to the parent node.
void print_node_post_order(const void *node) {
    BiTreeNode *treeNode = (BiTreeNode *) node;
    if (treeNode == NULL) return;
```

```
    print_node_post_order(treeNode->left);
    print_node_post_order(treeNode->right);
    printf("%d ", *((int *)treeNode->data));
}

// Remove all leaves in the tree.
// @Params: tree -- pointer to the tree.
void remove_leaves(BiTree *tree) {
    if (tree==NULL) return;
    remove_node_leaves(tree->root);
}

// Remove all leaf nodes following the current node.
// This is the helper function for @function{remove_leaves}.
// @Params: node -- pointer to current node.
void remove_node_leaves(BiTreeNode *node) {
    if (node == NULL) return;
    if (node->left != NULL) {
        if (bitree_is_leaf(node->left)) {
            free(node->left);
            node->left = NULL;
        }
        else remove_node_leaves(node->left);
    }
    if (node->right != NULL) {
        if (bitree_is_leaf(node->right)) {
            free(node->right);
            node->right = NULL;
        }
        else remove_node_leaves(node->right);
    }
}
```

Console Output:

```
Tree One Statistics:
------------------------------------
Total number of leaves:            3
Expected:                          3
Total number of non-leaf nodes:    6
Expected:                          6
Height of the tree:                5
Expected:                          5
Display in pre-order:
1 2 4 7 3 5 6 8 9
Expected:
1 2 4 7 3 5 6 8 9
Display in in-prder:
7 4 2 1 5 3 6 8 9
Expected:
7 4 2 1 5 3 6 8 9
Display in post-order:
7 4 2 5 9 8 6 3 1
Expected:
7 4 2 5 9 8 6 3 1
------------------------------------
Removing leaves...
------------------------------------
Remaining nodes:
1 2 4 3 6 8
Expected:
1 2 4 3 6 8


Tree Two Statistics:
------------------------------------
Total number of leaves:            5
Expected:                          5
Total number of non-leaf nodes:    4
Expected:                          4
```

```
Height of the tree:                4
Expected:                          4
Display in pre-order:
6 4 2 1 3 5 8 7 9
Expected:
6 4 2 1 3 5 8 7 9
Display in in-prder:
1 2 3 4 5 6 7 8 9
Expected:
1 2 3 4 5 6 7 8 9
Display in post-order:
1 3 2 5 4 7 9 8 6
Expected:
1 3 2 5 4 7 9 8 6
-----------------------------------
Removing leaves...
-----------------------------------
Remaining nodes:
6 4 2 8
Expected:
6 4 2 8


Program ended with exit code: 0
```