Student: Tran Ngoc Son （陳玉山）
ID: 0710185

Report homework 4

Contents

Student: Tran Ngoc Son （陳玉山）
ID: 0710185

I. Introduction

In this homework, we have to build a linked list binary tree data structure that has several functions. The functions are:

1. Tree-insert: to insert a node to a tree.
2. Tree print: to print out a binary tree.
3. Inorder-tree-walk: to print out an inorder array.
4. Tree-Minimum: to print out the smallest value of a tree.
5. Tree-maximum: to print out the largest value of a tree.
6. Tree-delete: to delete a node of a tree.
7. Tree-search: to search a node in a tree.

II. Methodology

1. Struct data.

- To create a node that has 4 variables, value, a pointer to left, a pointer to right, and a pointer to parent, we can make use of struct and define a Node same as below.

```
struct Node{
    int value;
    Node* left;
    Node* right;
    Node* parent;
};
```

Figure 1: A node structure

2. Binary tree

- A binary tree is a data structure that consists of 3 main parts, parent, left child, and right child. The left child of the parent is smaller than the parent itself, and the right child of the parent would be the one that has a larger value than its parent. The node that is first added to a tree would be called a root. Having a root is important since we can use it to retrieve all the data we need in a binary tree.

Student: Tran Ngoc Son （陳玉山）
ID: 0710185

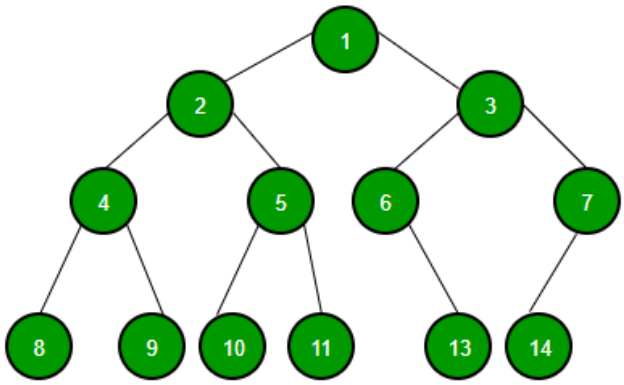

Figure 2: An example of a binary tree (Geekforgeek)

3. Tree print

- Since a tree with 50 nodes is large to plot out, therefore I captured the example tree with size of 10.

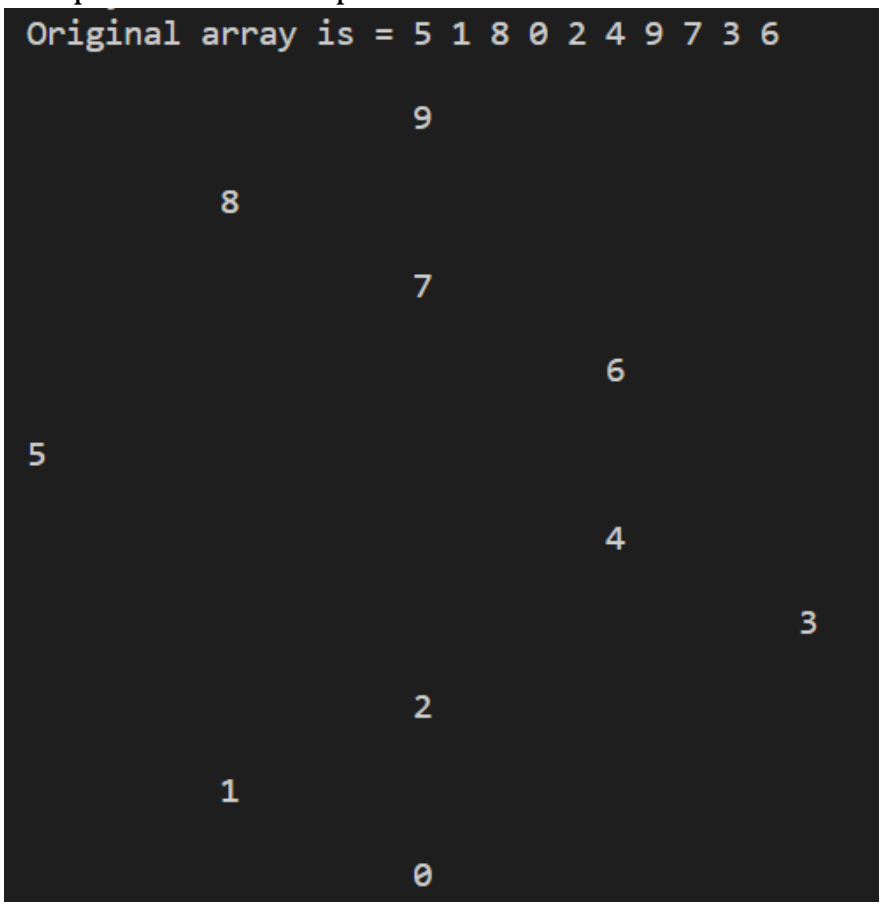

Figure 3: Binary tree plotting.

Student: Tran Ngoc Son （陳玉山）
ID: 0710185

4. Tree Minimum

- Taking advantage of the properties of the binary that the left child is always smaller than its parent, we can find the minimum value by keeping going to the left of a tree until it reaches the bottom.

```
The minimum is = 0
```

```
Node* tree_minimum(Node* root)
{
    Node* x = root;
    Node* y;
    while (x!=NULL)
    {   y = x;
        x = x->left;
    }
    cout << y->value;
    return y;

}
```

5. Tree maximum

- It is quite the same with the tree-minimum function. The right side's value is always larger than its parent, therefore we keep going right until it reaches the bottom to retrieve the maximum value of a tree.

```
The maximum is = 9
```

Student: Tran Ngoc Son （陳玉山）
ID: 0710185

```
Node* tree_maximum(Node* root)
{
    Node* x = root;
    Node* y;
    while (x!=NULL)
    {
        y = x;
        x = x->right;
    }
    cout << y->value;
    return y;
}
```

6. Inorder-walk-tree

- By using the recursive method, we can print out an inorder array of a tree same as below.
-

```
Inordered array is = 0 1 2 3 4 5 6 7 8 9
```

```
void inorder_tree_walk(Node* root)
{
    Node* x = root;
    if (x == NULL)
        return;

    inorder_tree_walk(x->left);
    cout << x->value << " ";
    inorder_tree_walk(x->right);
    // cout << x -> value << " ";
}
```

Student: Tran Ngoc Son （陳玉山）
ID: 0710185

7. Tree inserting

- This is an import function that we need to create a binary tree. It has to find the proper position to put a new node.

```
Node* Tree_insert(Node* root, Node* new_node)
{
    if (root == NULL)
    {
        root = new_node;
        new_node -> parent = NULL;
    }
    else
    {
        Node* x = root;
        Node* y; //this one is to track the parent node
        while (x!=NULL)
        {
            y = x;
            if (new_node->value <= x->value)
                x = x->left;
            else x = x -> right;
        }
        new_node -> parent = y;
        if (new_node -> value < y->value)
            y -> left = new_node;
        else y -> right = new_node;

    }
    return root;
}
```

8. Tree deleting

- When deleting a node, we first have to find out the address of that node and then call the tree deleting function to delete it. In the tree delete function, we need to cover all the circumstances that could happen when deleting a node.

Student: Tran Ngoc Son （陳玉山）
ID: 0710185

```
Array after deleting 6
0 1 2 3 4 5 7 8 9

                    9

          8

                    7

5

                              4

                                        3

                    2

          1

                    0
```

9. Search function

- To implement this function, we would use the comparison between the value of the current node with the value we want to search for. If a searching value is larger than the current node's value, we then go right and go left by contrast.

```
Node* search(Node* root, int data)
{
    Node* x = root;

    while (x!= NULL)
    {
        if (x->value == data)
            return x;
        else if (x->value > data)
            x = x->left;
        else x = x -> right;
    }
    return 0;
}
```

Student: Tran Ngoc Son （陳玉山）
ID: 0710185

III. Discussion

- I think the most complicated part of this homework is the deleting part since we have to cover all the possibilities that could happen when deleting a node. After working on this homework, I feel more comfortable with the pointer and have a deeper understanding of the Binary tree.

IV. Conclusion

- A binary tree is a very popular structure in programming, in many cases, it can help to decrease the time of searching, sorting, etc.