Assignments week 4

Ethan Bastian, Abdullah Turk

2025-09-28

Assignment 01 - maintaining a sorted tree

My code:

```
// fancy algorithm here

#include <algorithm>
#include <cinttypes>
#include <cstdio>
#include <iostream>
#include <string>
#include "bintree.h"

sax::binary_tree_node<int> * insert (sax::binary_tree_node<int> * root, int value) {

    if(root == nullptr) {
        return new sax::binary_tree_node<int>{value, nullptr, nullptr};
    }

    if (value < root->data) {
        root->left = insert(root->left, value);
    } else if (value > root->data) {
        root->right = insert(root->right, value);
    } else {
        return root;
    }

    return root;
}

sax::binary_tree_node<int> * find_min(sax::binary_tree_node<int> * root) {
    while (root->left != nullptr) {
        root = root->left;
    }
    return root;
}

sax::binary_tree_node<int> * remove (sax::binary_tree_node<int> * root, int value) {



    if (root==nullptr) {
        return nullptr;
    }
    if (value < root->data) {
        root->left = remove(root->left, value);
    } else if (value > root->data) {
        root->right = remove(root->right, value);
    } else {
        if(root->right == nullptr && root->left == nullptr){
            delete root;
            return nullptr;
        }
        if (root->right == nullptr) {
            sax::binary_tree_node<int> * temp = root->left;
            delete root;
            return temp;
        }
        if (root->left == nullptr) {
            sax::binary_tree_node<int> * temp = root->right;
            delete root;
            return temp;
        }

        auto min_node = find_min(root->right);
        root->data = min_node->data;
        root->right = remove(root->right, min_node->data);


    }


    return root;
}

int main() {
    /* TODO:
    Write a program that reads a binary search tree with integers from the standard input,
    followed by a command to either insert or delete an integer from the tree.
    The program should then perform the specified operation and print the resulting tree.
    */

    sax::binary_tree_node<int>* root = nullptr;
    std::string command;
    int value;
    std::cin >> root >> command >> value;

    if (command == "insert") {
        root = insert(root,value);
    } else if (command == "remove") {
        root = remove(root,value);
    }

    std::cout << root << std::endl;
    sax::binary_tree_node<int>::cleanup(root);
    return 0;
}
```

Time complexity: this algorithm has a time complexity of . . . .., because . . . . . .

Assignment 2 - tree slicing

My code:

```
// fancy algorithm here

#include <iostream>
#include "bintree.h"
#include "node_ptr.h"



int main()
{
    /* TODO:
        Write a program that reads a binary search tree from the standard input,
        and prints the values in the tree in reverse sorted order.

        Example input: ((1 3 4) 5 (7 8 ()))
        Example output: 8 7 5 4 3 1

        Use the binary_tree_node struct from bintree.h to represent the tree. This structur
        contains input and output operators that you can use to read (and write) trees.
    */

    sax::binary_tree_node<int> * input = nullptr;
    int k;
    std::cin >> input >> k;

    sax::node_ptr<int> it(input);
    bool first = true;

    while(it && it->data <= k) {
        if (!first) {
            std::cout << " ";

        }
        std::cout << it->data;
        first = false;
        it.move_next();
    }

    std::cout << std::endl;
    sax::binary_tree_node<int>::cleanup(input);

    return 0;
}
```

and here is the node_ptr

```
#ifndef NODE_PTR_H
#define NODE_PTR_H

#include <stack>    // for std::stack
#include "bintree.h"

namespace sax {
    template<typename T>
    class node_ptr {
    public:
        node_ptr(binary_tree_node<T>* root) {
            push_left_path(root);
        }
        binary_tree_node<T>* operator->() {
            return st.top();
        }

        operator bool() const {

            return !st.empty();
        }
        void move_next() {
            auto node = st.top();
            st.pop();
            if (node->right) {
                push_left_path(node->right);
            }
        }
        node_ptr& operator++() {
            move_next();
            return *this;
        }
```

Time complexity: this algorithm has a time complexity of . . . .., because . . . . . .

Assignment 3 - Subs, Sets and Trees

My code:

```
// fancy algorithm here

#include <iostream>
#include <string>
#include "bintree.h" // for binary_tree_node
#include "node_ptr.h"

bool is_subset(sax::binary_tree_node<int>*one, sax::binary_tree_node<int>*two) {
    sax::node_ptr<int> itone(one);
    sax::node_ptr<int> ittwo(two);

    while (itone && ittwo) {
        if (itone->data < ittwo->data) {
            return false;
        } else if (itone->data > ittwo->data) {
            ++ittwo;
        } else {
            ++itone;
            ++ittwo;
        }
    }

    return !itone;
}
```

```
int main() {
    /* TODO:
        Write a program that reads two binary search trees from standard input, which repre
        integers (no duplicates within each tree), and checks if the first one is a subset o

        The program must output "true" if the first tree is a subset of the second one, and

        The time complexity of your program must be O(n + m), where n and m are the sizes o
    */

    sax::binary_tree_node<int> * one = nullptr;
    sax::binary_tree_node<int> * two = nullptr;

    std::cin >> one >> two;

    sax::node_ptr<int> itone(one);
    sax::node_ptr<int> ittwo(two);


    bool result = is_subset(one,two);
    std::cout << (result ? "true" : "false") << std::endl;

    sax::binary_tree_node<int>::cleanup(one);
    sax::binary_tree_node<int>::cleanup(two);


    return 0;
}
```

Time complexity: this algorithm has a time complexity of . . . .., because . . . . . .