Міністерство освіти і науки України Львівський національний університет імені Івана Франка Факультет електроніки та комп'ютерних технологій

Звіт

про виконання лабораторної роботи №4

“Дерева. Бінарні дерева та бінарні дерева пошуку (BST)**”**

Виконав:
студент 2 курсу
групи ФЕП-23с
Чепара Станіслав
Викладач:
доц. Середницька Христина Ігорівна

Львів - 2025

Мета роботи

Ознайомитися зі структурою бінарних дерев, навчитися будувати звичайне бінарне дерево фіксованого розміру,

виконувати основні обходи (Preorder, Inorder, Postorder), а також реалізувати бінарне дерево пошуку (BST)

з операціями пошуку, вставки, видалення, знаходження мінімуму/максимуму, наступника та попередника.

Теоретичні відомості

• Вузол (Node): містить ключ та посилання на лівого/правого нащадків і на батька.

• Бінарне дерево: у кожного вузла не більше двох дітей (left, right).

• Обходи дерева:

– Preorder (Prefix): вузол → ліво → право.

– Inorder (Infix): ліво → вузол → право (у BST дає відсортований вивід).

– Postorder (Postfix): ліво → право → вузол.

• BST: ліве піддерево містить ключі < вузол; праве піддерево містить ключі > вузол. Дублікатів не зберігаємо.

Постановка задачі

Частина 1. Реалізувати звичайне бінарне дерево заданого розміру n, функції CreateTree, ShowTree (красиве відображення), та обходи In/Pre/Post.

Частина 2. Реалізувати BST з функціями Search, Insert, Delete, Minimum, Maximum, Successor, Predecessor, а також перевірити сортування через Inorder.

Структура проєкту

include/Tree.h

src/Tree.cpp

src/Lab_04_01.cpp # звичайне дерево: створення/показ/обходи

src/Lab_04_02.cpp # BST: пошук/вставка/видалення/наступник/попередник + InOrder

ФРАГМЕНТИ КОДУ

Tree.h

```cpp
#pragma once
#include <iostream>
#include <vector>

using datatype = int;

struct Node {
    datatype key;
    Node* parent;
    Node* left;
    Node* right;
    explicit Node(datatype k) : key(k), parent(nullptr), left(nullptr), right(nullptr) {}
};

// ===== Частина 1: звичайне бінарне дерево =====
void CreateTree(Node*& root, int n,
                const std::vector<datatype>* keys = nullptr,
                int* idx = nullptr);
// Відображення дерева (красиве ASCII): праве вгорі, ліве внизу
void ShowTree(const Node* root, int L = 0);
void PreOrder(const Node* root);
void InOrder(const Node* root);
void PostOrder(const Node* root);
void FreeTree(Node*& root);

// ===== Частина 2: BST =====
Node* CreateRootBST(datatype key);
Node* SearchNodeBST(Node* root, datatype key);
Node* InsertNodeBST(Node*& root, datatype key); // nullptr якщо дублікат
Node* MinimumNodeBST(Node* root);
Node* MaximumNodeBST(Node* root);
Node* SuccessorNodeBST(Node* pNode);
Node* PredecessorNodeBST(Node* pNode);
bool  DeleteNodeBST(Node*& root, Node* delNode);

```

Tree.cpp

```cpp
static void buildTree(Node*& root, int n, const std::vector<datatype>& keys, int& idx) {
    if (n <= 0) { root = nullptr; return; }
    root = new Node(keys[idx++]);
    int n_left  = n / 2;
    int n_right = n - n_left - 1;
    buildTree(root->left,  n_left,  keys, idx);
    if (root->left)  root->left->parent  = root;
    buildTree(root->right, n_right, keys, idx);
    if (root->right) root->right->parent = root;
}

// ================ Частина 1: звичайне бінарне дерево ================

void CreateTree(Node*& root, int n, const std::vector<datatype>* keys, int* idx) {
    if (n <= 0) { root = nullptr; return; }
    if (keys && idx) {
        int localIdx = *idx;
        buildTree(root, n, *keys, localIdx);
        *idx = localIdx;
        return;
    }
    std::vector<datatype> autoKeys(n);
    for (int i = 0; i < n; ++i) autoKeys[i] = i + 1;
    int localIdx = 0;
    buildTree(root, n, autoKeys, localIdx);
}


static void showPretty(const Node* node, const std::string& prefix, bool isLeft) {
    if (!node) return;
    if (node->right) showPretty(node->right, prefix + (isLeft ? "│   " : "    "), false);
    std::cout << prefix << (isLeft ? "└── " : "┌── ") << node->key << '\n';
    if (node->left)  showPretty(node->left,  prefix + (isLeft ? "    " : "│   "), true);
}

void ShowTree(const Node* root, int /*L*/) {
    showPretty(root, "", true);
}

void PreOrder(const Node* p) { //(Prefix)обхід у прямому порядку(вузол → ліво → право)
    if (!p) return;
    std::cout << p->key << ' ';
    PreOrder(p->left);
    PreOrder(p->right);
}

void InOrder(const Node* p) { //(Infix)обхід у внутрішньому порядку:(ліво → вузол → право) 1 2 3 4 5 для BST завжди дає відсортовані ключ
    if (!p) return;
    InOrder(p->left);
    std::cout << p->key << ' ';
    InOrder(p->right);
}

void PostOrder(const Node* p) { //(Postfix)обхід у зворотньому порядку: (ліво → право → вузол)
    if (!p) return;
    PostOrder(p->left);
    PostOrder(p->right);
    std::cout << p->key << ' ';
}

void FreeTree(Node*& root) {
    if (!root) return;
    FreeTree(root->left);
    FreeTree(root->right);
    delete root;
    root = nullptr;
}
```

```cpp
// ================ Частина 2: BST ================

Node* CreateRootBST(datatype key) { return new Node(key); }

Node* SearchNodeBST(Node* root, datatype key) {
    Node* p = root;
    while (p) {
        if (key == p->key) return p;
        p = (key < p->key) ? p->left : p->right;
    }
    return nullptr;
}

Node* InsertNodeBST(Node*& root, datatype key) {
    if (!root) { root = CreateRootBST(key); return root; }
    Node* p = root; Node* prev = nullptr;
    while (p) {
        prev = p;
        if (key == p->key) return nullptr;
        p = (key < p->key) ? p->left : p->right;
    }
    Node* nw = new Node(key);
    nw->parent = prev;
    if (key < prev->key) prev->left = nw; else prev->right = nw;
    return nw;
}

Node* MinimumNodeBST(Node* root) {
    if (!root) return nullptr;
    Node* p = root;
    while (p->left) p = p->left;
    return p;
}

Node* MaximumNodeBST(Node* root) {
    if (!root) return nullptr;
    Node* p = root;
    while (p->right) p = p->right;
    return p;
}

Node* SuccessorNodeBST(Node* x) {
    if (!x) return nullptr;
    if (x->right) return MinimumNodeBST(x->right);
    Node* p = x->parent;
    while (p && x == p->right) { x = p; p = p->parent; }
    return p;
}

Node* PredecessorNodeBST(Node* x) {
    if (!x) return nullptr;
    if (x->left) return MaximumNodeBST(x->left);
    Node* p = x->parent;
    while (p && x == p->left) { x = p; p = p->parent; }
    return p;
}

bool DeleteNodeBST(Node*& root, Node* del) {
    if (!del) return false;
    if (del->left && del->right) {
        Node* succ = SuccessorNodeBST(del);
        del->key = succ->key;
        del = succ;
    }
    Node* child = del->left ? del->left : del->right;
    if (child) child->parent = del->parent;
    if (!del->parent) {
        root = child;
    } else if (del == del->parent->left) {
        del->parent->left = child;
    } else {
        del->parent->right = child;
```

```cpp

bool DeleteNodeBST(Node*& root, Node* del) {
    if (!del) return false;
    if (del->left && del->right) {
        Node* succ = SuccessorNodeBST(del);
        del->key = succ->key;
        del = succ;
    }
    Node* child = del->left ? del->left : del->right;
    if (child) child->parent = del->parent;
    if (!del->parent) {
        root = child;
    } else if (del == del->parent->left) {
        del->parent->left = child;
    } else {
        del->parent->right = child;
    }
    delete del;
    return true;
}

```

Lab_04_01.cpp

```cpp
#include "Tree.h"
using std::cout; using std::cin; using std::endl;

int main() {
    Node* root = nullptr;
    while (true) {
        cout << "Виберіть функцію:  1) CreateTree  2) ShowTree  3) InfixOrder  4) PostfixOrder  5) PrefixOrder  6) Exit\n> ";
        int op; if (!(cin >> op)) return 0;
        if (op == 6) break;
        if (op == 1) {
            FreeTree(root);
            cout << "Введіть розмір дерева: ";
            int n; cin >> n;
            cout << "Ввести власні ключі? (1=так, 0=ні): ";
            int own; cin >> own;
            if (own) {
                std::vector<datatype> keys(n);
                for (int i = 0; i < n; ++i) { cout << "Введіть ключ: "; cin >> keys[i]; }
                int idx = 0;
                CreateTree(root, n, &keys, &idx);
            } else {
                CreateTree(root, n, nullptr, nullptr);
            }
            cout << "Tree has been created." << endl;
        } else if (op == 2) {
            ShowTree(root);
        } else if (op == 3) {
            InOrder(root); cout << '\n';
        } else if (op == 4) {
            PostOrder(root); cout << '\n';
        } else if (op == 5) {
            PreOrder(root); cout << '\n';
        }
    }
    FreeTree(root);
    return 0;
}

```

Вивід:

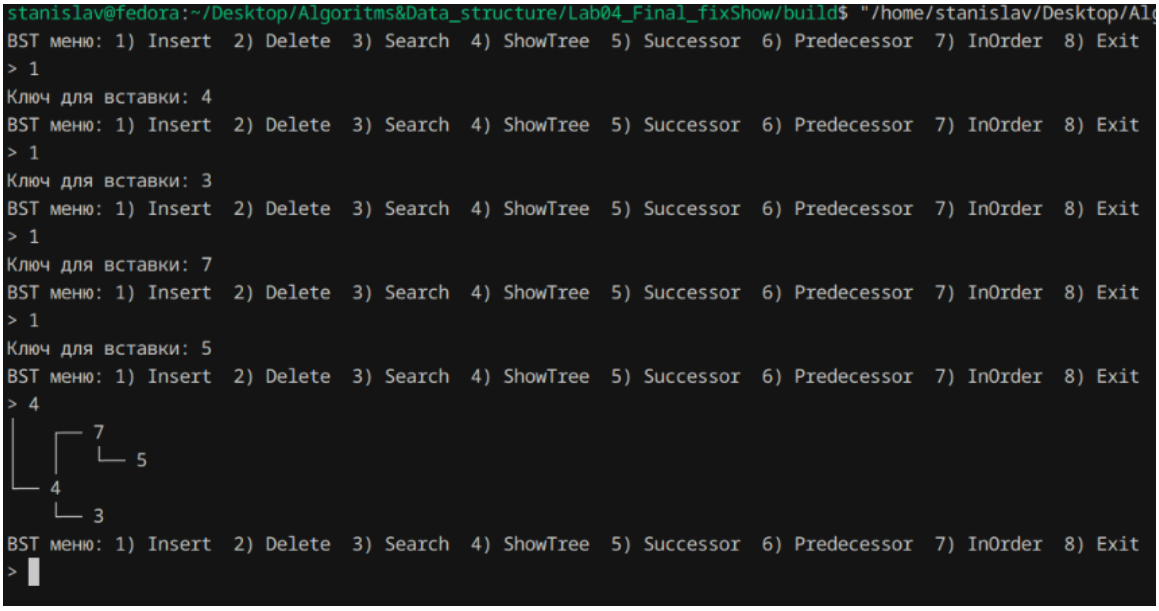
```
stanislav@fedora:~/Desktop/Algoritms&Data_structure/Lab04_Final_fixShow/build$ "/home/stanislav/Desktop/
Виберіть функцію:  1) CreateTree  2) ShowTree  3) InfixOrder  4) PostfixOrder  5) PrefixOrder  6) Exit
> 1
Введіть розмір дерева: 6
Ввести власні ключі? (1=так, 0=ні): 0
Tree has been created.
Виберіть функцію:  1) CreateTree  2) ShowTree  3) InfixOrder  4) PostfixOrder  5) PrefixOrder  6) Exit
> 2
    ┌── 5
    │   └── 6
└── 1
    │   ┌── 4
    └── 2
        └── 3
Виберіть функцію:  1) CreateTree  2) ShowTree  3) InfixOrder  4) PostfixOrder  5) PrefixOrder  6) Exit
>
```

Lab_04_02.cpp

```cpp
#include "Tree.h"
using std::cout; using std::cin; using std::endl;

int main() {
    Node* root = nullptr;
    while (true) {
        cout << "BST меню: 1) Insert  2) Delete  3) Search  4) ShowTree  5) Successor  6) Predecessor  7) InOrder  8) Exit\n> ";
        int op; if (!(cin >> op)) return 0;
        if (op == 8) break;
        if (op == 1) {
            cout << "Ключ для вставки: "; datatype k; cin >> k;
            auto res = InsertNodeBST(root, k);
            if (!res) cout << "Дублікат: такий ключ вже існує." << endl;
        } else if (op == 2) {
            cout << "Ключ для видалення: "; datatype k; cin >> k;
            Node* t = SearchNodeBST(root, k);
            if (!t) cout << "Не знайдено." << endl;
            else { DeleteNodeBST(root, t); cout << "Видалено." << endl; }
        } else if (op == 3) {
            cout << "Ключ для пошуку: "; datatype k; cin >> k;
            Node* t = SearchNodeBST(root, k);
            cout << (t ? "Знайдено." : "Не знайдено.") << endl;
        } else if (op == 4) {
            ShowTree(root);
        } else if (op == 5) {
            cout << "Ключ для Successor: "; datatype k; cin >> k;
            Node* t = SearchNodeBST(root, k);
            if (!t) cout << "Не знайдено вузол." << endl;
            else {
                Node* s = SuccessorNodeBST(t);
                if (s) cout << "Successor: " << s->key << endl; else cout << "Немає (максимальний елемент)." << endl;
            }
        } else if (op == 6) {
            cout << "Ключ для Predecessor: "; datatype k; cin >> k;
            Node* t = SearchNodeBST(root, k);
            if (!t) cout << "Не знайдено вузол." << endl;
            else {
                Node* p = PredecessorNodeBST(t);
                if (p) cout << "Predecessor: " << p->key << endl; else cout << "Немає (мінімальний елемент)." << endl;
            }
        } else if (op == 7) {
            InOrder(root); cout << '\n';
        }
    }
    FreeTree(root);
    return 0;
}
```

Вивід:



ВИСНОВОК:У ході лабораторної роботи було реалізовано алгоритми створення та обходу бінарного дерева, а також основні операції з бінарним деревом пошуку: пошук, вставку, видалення та визначення вузлів-наступників і попередників. Отримано практичні навички роботи з рекурсивними структурами даних у мові C/C++, що дозволяє ефективно організовувати зберігання та обробку інформації у вигляді дерев.