**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**Львівський національний університет імені Івана Франка**

**Факультет електроніки та комп’ютерних технологій**

**ЗВІТ**

**З ЛАБОРАТОРНОЇ РОБОТИ № 1**

з дисципліни „Прикладна статистика та ймовірнісні процеси”

на тему **„** **”**

Виконав

студент 2 курсу

групи ФеП-21

Перевірив

доцент кафедри РКТ, к.ф.-м.н.

**Сас Наталія Богданівна**

**Львів – 2022**

**Мета роботи:** Опанувати теорію та написати програму для побудови та обходу бінарного дерева. Бінарного дерева пошуку, та необхідних для його роботи функцій.

**Теоретичні відомості:**

Деревомназивається зв’язний граф без простих циклів (це одне з багатьох означень)

Деревоможна також означити як рекурсивну структуру вузлів, яка є або порожньою, або складається з базового вузла –кореня

Якщо дві вершини у дереві суміжні, то одну з них називають батьком, а іншу сином. За необхідності, ребро від батька до сина можна вважати орієнтованим. Вершина, що не має батька називається коренем дерева. Вершини, що не мають синів називаються листками. Вершини, які мають синів називаються внутрішніми(корінь також належить до внутрішніх вершин). Якщо існує шлях (орієнтований) з однієї вершини дерева в іншу, то одна з цих вершин є предком, а інша –нащадком(батько і син є частковим випадком). Нехай вершина не є листком. Тоді підграф, що містить цю вершину і всіх її нащадків називається піддеревом із коренем в цій вершині.

Кількість синів вершини νназивають її степенем(позн. deg( )v). Дерево називається m-арним, якщо жодна вершина має не більше ніж m синів (deg( )vm≤). Якщо кожна вершина має точно m синів(deg( )vm=), то дерево називається повнимm-арним.

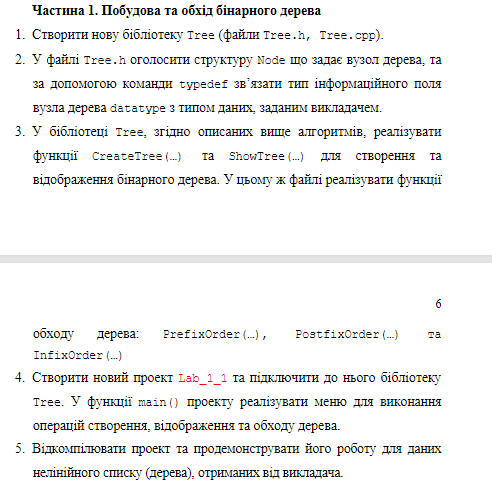
Рівнем вершини L називається довжина шляху від кореня до цієї вершини (рівень кореня рівний нулеві).Висотою дерева H називають довжину найдовшого шляху між вершинами в ньому (тобто максимальний рівень). Дерево називається збалансованим, якщо всі його листки знаходяться на одному, або сусідніх рівнях(тобто рівнях Hта H–1).

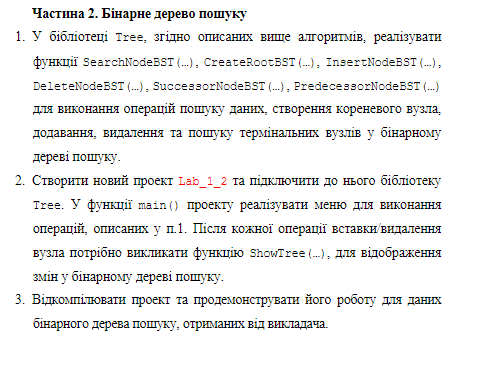
Бінарним деревом пошуку(англ. binarysearchtree(BST)) називається бінарне дерево, що задовольняєтаким умовам:

•поля даних (ключі) у різних вузлах не можуть бути однакові;

•для будь-якого вузла поля даних (ключі) вузлів його лівого піддерева є меншими, а правого –більшими за ключ цього вузла.

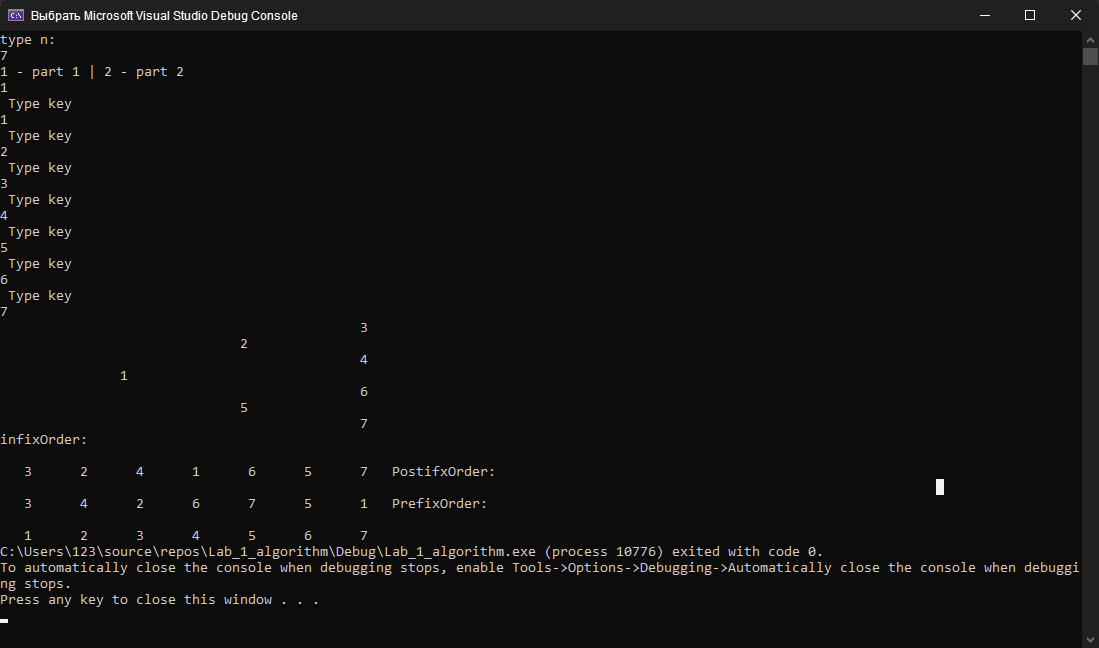
**Зміст роботи:**



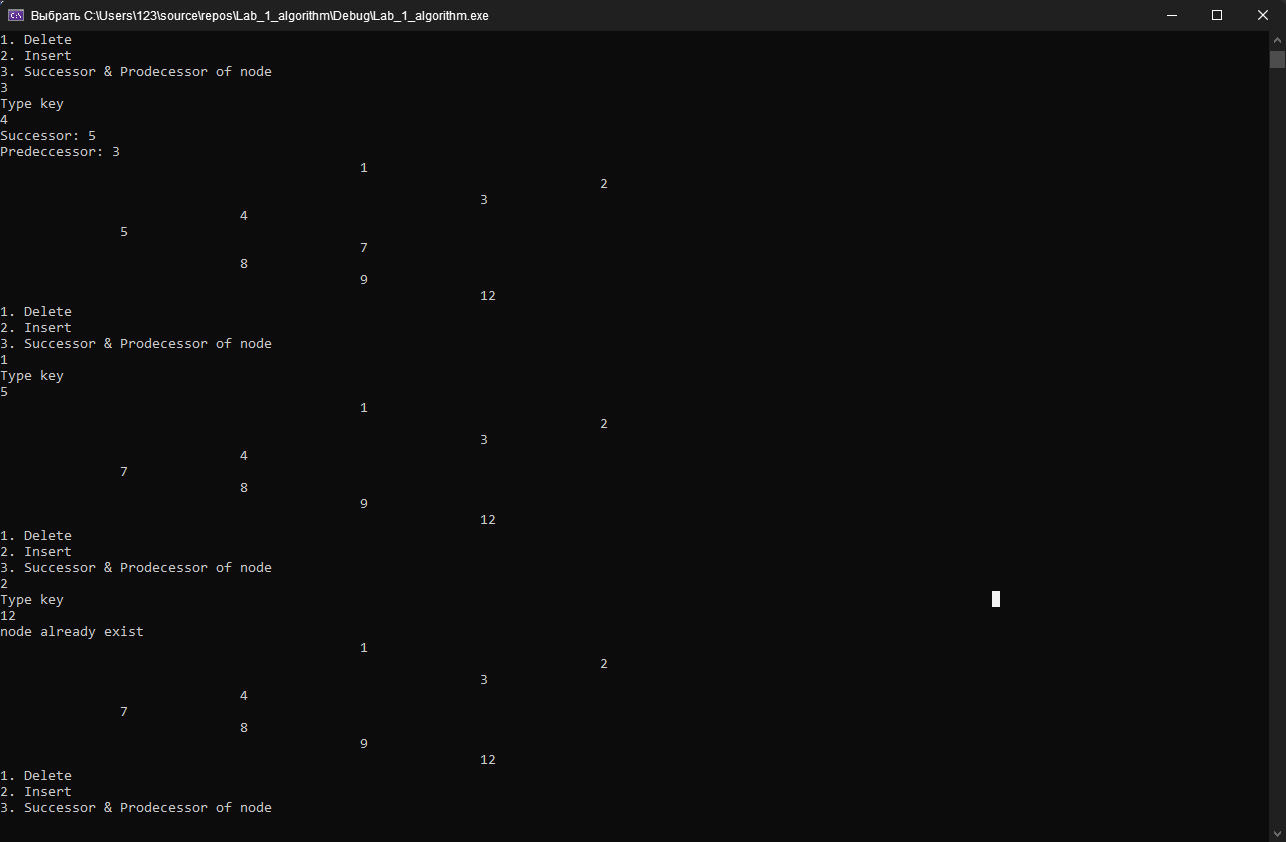


**Результати:**

Частина 1:



Частина 2:



**Висновки:**

На цій лабораторній роботі я ознайомився із теорією побудови та обходу бінарних дерев (пошуку). Навчився за допомогою мови програмування С++ реалізовувати функції для побудови та обходу дерев, видалення, вставки елементів, знаходження наступного та попереднього елементів.

**Додаток:**

**Вміст файлу Tree.h:**

#pragma once

#include <cstddef>

struct Node {

int key;

Node\* parent;

Node\* left;

Node\* right;

Node\* previous;

void createTree(int data, Node\*& pNode,int n);

void showTree(Node\* pNode, int L);

void prefixOrder(Node\*& pNode);

void postfixOrder(Node\*& pNode);

void infixOrder(Node\*& pNode);

Node\* SearchNodeBST(int data, Node\* pNode,int k);

void createRoot(int data, Node\*& pNode);

int insertNode(int data, Node\*& pNode);

Node\* delNode(int data, Node\*& pNode);

Node\* SuccessorNodeBST(Node\* pNode,int key);

Node\* PredeccessorNodeBST(Node\* pNode, int key);

Node\* maximum(Node\* pNode);

Node\* minimum(Node\* pNode);

};

**Вміст файлу Tree.cpp:**

#include "Tree.h"

#include <iostream>

using namespace std;

int n\_left;

int n\_right;

void Node::createTree(int data, Node\*& pNode,int n) {

if (n == 0) {

pNode = NULL;

}

else {

cout << " Type key " << endl;

cin >> data;

pNode = new Node;

pNode->key = data;

pNode->left = NULL;

pNode->right = NULL;

createTree(data, pNode->left, n / 2);

createTree(data, pNode->right, n - n/2 - 1);

}

}

void Node::showTree(Node\* pNode, int L) {

if (pNode)

{

showTree(pNode->left, L + 5);

for (int i = 0; i < L; i++) cout << " ";

cout << pNode->key << endl;

showTree(pNode->right, L + 5);

}

}

void Node::prefixOrder(Node\*& pNode) {

if (pNode != NULL) {

cout << " " << pNode->key << " ";

prefixOrder(pNode->left);

prefixOrder(pNode->right);

}

}

void Node::postfixOrder(Node\*& pNode) {

if (pNode != NULL) {

postfixOrder(pNode->left);

postfixOrder(pNode->right);

cout << " " << pNode->key << " ";

}

}

void Node::infixOrder(Node\*& pNode) {

if (pNode != NULL) {

infixOrder(pNode->left);

cout << " " << pNode->key << " ";

infixOrder(pNode->right);

}

}

/////////////////////////////

Node\* Node::SearchNodeBST(int data, Node\* pNode,int k) {

if (pNode != NULL) {

if (pNode->key == data) {

if (k == 1) {

SuccessorNodeBST(pNode, data);

PredeccessorNodeBST(pNode, data);

}

return pNode;

}

else

if(pNode->key > data){

SearchNodeBST(data, pNode->left,k);

}

else {

SearchNodeBST(data, pNode->right,k);

}

}

else {

return NULL;

}

}

void Node::createRoot(int data, Node\*& pNode) {

if (!pNode) {

pNode = new Node;

pNode->key = data;

pNode->left = NULL;

pNode->right = NULL;

pNode->parent = NULL;

return;

}

}

int Node::insertNode(int data, Node\*& pNode) {

if (SearchNodeBST(data, pNode,0) != NULL) {

cout << "node already exist" << endl;

return 0;

}

else if (!pNode) {

pNode = new Node;

pNode->key = data;

pNode->left = NULL;

pNode->right = NULL;

pNode->parent = NULL;

return 1;

}

if (pNode != NULL) {

previous = pNode;

if (pNode->key > data)

{

insertNode(data, pNode->left);

}

else

{

insertNode(data, pNode->right);

}

}

return 1;

}

Node\* Node::delNode(int data, Node\*& pNode) {

if (pNode == NULL)

return pNode;

if (data < pNode->key)

pNode->left = delNode(data, pNode->left);

else if (data > pNode->key)

pNode->right = delNode(data, pNode->right);

else if (pNode->left != NULL && pNode->right != NULL) {

pNode->key = minimum(pNode->right)->key;

pNode->right = delNode(pNode->key, pNode->right);

}

else {

if (pNode->left != NULL)

pNode = pNode->left;

else if (pNode->right != NULL)

pNode = pNode->right;

else

pNode = NULL;

}

return pNode;

}

Node\* Node::SuccessorNodeBST(Node\* pNode, int key) {

Node\* current = this; Node\* successor = NULL;

while(current != NULL) {

if (current->key > key){

successor = current;

current = current->left;

}

else {

current = current->right;

}

}

if (successor != NULL) {

cout << "Successor: " << successor->key << endl;

return successor;

}

else

cout << " Successor is not exist! " << endl;

}

Node\* Node::PredeccessorNodeBST(Node\* pNode,int key) {

Node\* current = this; Node\* successor = NULL;

while (current != NULL) {

if (current->key < key) {

successor = current;

current = current->right;

}

else {

current = current->left;

}

}

if (successor != NULL) {

cout << "Predeccessor: " << successor->key << endl;

return successor;

}

else

cout << " Predeccessor is not exist! " << endl;

}

Node\* Node::maximum(Node\* pNode) {

if (pNode->right == NULL)

return pNode;

maximum(pNode->right);

}

Node\* Node::minimum(Node\* pNode) {

if (pNode->left == NULL)

return pNode;

minimum(pNode->left);

}

**Вміст файлу головної програми:**

#include "Tree.h"

#include <iostream>

using namespace std;

int main()

{

int n, num, var,k;

Node\* pNode = NULL;

Node\* pNode2 = NULL;

int numbs[100];

cout << "type n: " << endl;

cin >> n;

cout << "1 - part 1 | 2 - part 2" << endl;

cin >> var;

switch (var) {

case 1: {

numbs[0] = 0;

pNode->createTree(numbs[0], pNode, n);

pNode->showTree(pNode, 5);

cout << "infixOrder:\n" << endl;

pNode->infixOrder(pNode);

cout << "PostifxOrder:\n" << endl;

pNode->postfixOrder(pNode);

cout << "PrefixOrder:\n" << endl;

pNode->prefixOrder(pNode);

} break;

case 2: {

for (int i = 0; i < n; i++) {

cout << " num" << i << ":" << endl;

cin >> numbs[i];

if (i!=0)

pNode->insertNode(numbs[i], pNode);

else

pNode->createRoot(numbs[i], pNode);

}

mark1:

pNode->showTree(pNode, 5);

cout << "1. Delete\n" << "2. Insert\n" << "3. Successor & Prodecessor of node" << endl;

cin >> var;

switch (var) {

case 1: {

cout << "Type key" << endl;

cin >> num;

pNode->delNode(num, pNode);

}break;

case 2: {

cout << "Type key" << endl;

cin >> num;

pNode->insertNode(num, pNode);

}break;

case 3: {

cout << "Type key" << endl;

cin >> num;

pNode->SearchNodeBST(num, pNode,1);

}

}

goto mark1;

} break;

}

}