МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ



**Дніпровський національний університет  
залізничного транспорту імені академіка В. Лазаряна**

Кафедра «Комп'ютерні інформаційні технології»

**Лабораторна робота №12**

**з дисципліни «Основи програмування»**

**на тему: «**Бінарне дерево пошуку**»**

Виконав:

студент гр.ПЗ1911

Сафонов Д.Є.

Прийняла:

Нежуміра О. І.

Дніпро, 2020

**Тема.** Бінарне дерево пошуку.

**Мета.** Засвоїти поняття ієрархічної динамічної структури. Оволодіти навичками створення бінарного дерева пошуку та виконання операцій над його елементами. Засвоїти рекурсивні методи аналізу бінарних дерев пошуку.

# **Постановка завдання (завдання до лабораторної роботи, індивідуальне завдання).**

Розробити програму для заданого індивідуального завдання і обраного рівня складності для роботи з бінарними деревами пошуку. Структура вершини дерева містить такі поля: ключ, дані и два вказівника на піддерева – ліве і праве. Якщо піддерево відсутнє, вказівник дорівнює NULL. Дерево може мати вершини з однаковими ключами, але не може мати вершини з однаковими ключами і даними одночасно. При розробці функцій слід використовувати механізм рекурсії.

Вимоги до програми:

− валідація вхідних даних;

− програма повинна мати багатофайлову, яка базується на реалізації функціонала програми.

Вимоги до тексту програми:

− специфікації програми і її функцій;

− самодокументованість коду: всі ідентифікатори повинні мати назви, що відповідають суті змінних.

Рівень III (82-89 балів, дуже добре). Реалізувати функції:

1) додавання вершини до бінарного дерева пошуку по ключу,

2) друку вершин дерева за заданим порядком в форматі «(ключ, дані)»,

3) видалення вершини по ключу (першого знайденого),

4) видалення всього дерева,

5) створення бінарного дерева пошуку шляхом читання інформації з текстового файлу,

6) створення бінарного дерева пошуку шляхом читання інформації з бінарного файлу,

7) запису бінарного дерева пошуку в такому ж порядку і формати, як і для друку, в текстовий файл,

8) запису бінарного дерева пошуку в такому ж порядку, як і для друку, в бінарний файл,

9) «дія» з Таблиці:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Ключ | Дані | Дія | Друк вершин дерева |
| 1 | Рік видання | Автор(и) , назва книги | Обхід дерева по рівням | В зворотному порядку |

# **2. Зовнішні специфікації програми (вхідні та вихідні дані, функціональні вимоги до програми).**

Програма буде розроблена як тест розробленого функціоналу, тож вхідні дані будуть одразу задані, але не дивлячись на це будуть наведені приклади(аргументи публічних методів та функцій).

2.1. Вхідні дані:

|  |  |  |  |
| --- | --- | --- | --- |
| Найменування даних | Умовне позначення | Вимоги до данних | Приклад |
| Ключ пошуку | key | Дійсне значення типу KEY | Якщо KEY – int, то 62 |
| Тип обходу | traversal | preOrder, inOrder, postOrder | inOrder |
| Функція – тіло циклу обходу дерева | func | void(DATA&, key) | {} |
| Рівень дерева | level | Натуральне число меньше або рівне висоті дерева | 1 |

2.2. Вихідні дані:

|  |  |  |  |
| --- | --- | --- | --- |
| Найменування даних | Умовне позначення | Вимоги до данних | Приклад |
| висота дерева | height | Кількість рівнів дерева | 1 |

2.3. Функціональні вимоги:

− валідація вхідних даних(аргументів функцій);

− програма повинна мати багатофайлову, яка базується на реалізації функціонала програми.

* Перевірка усіх функцій зазначених у завданні.

# **3. Опис структур даних, використаних для реалізації завдання.**

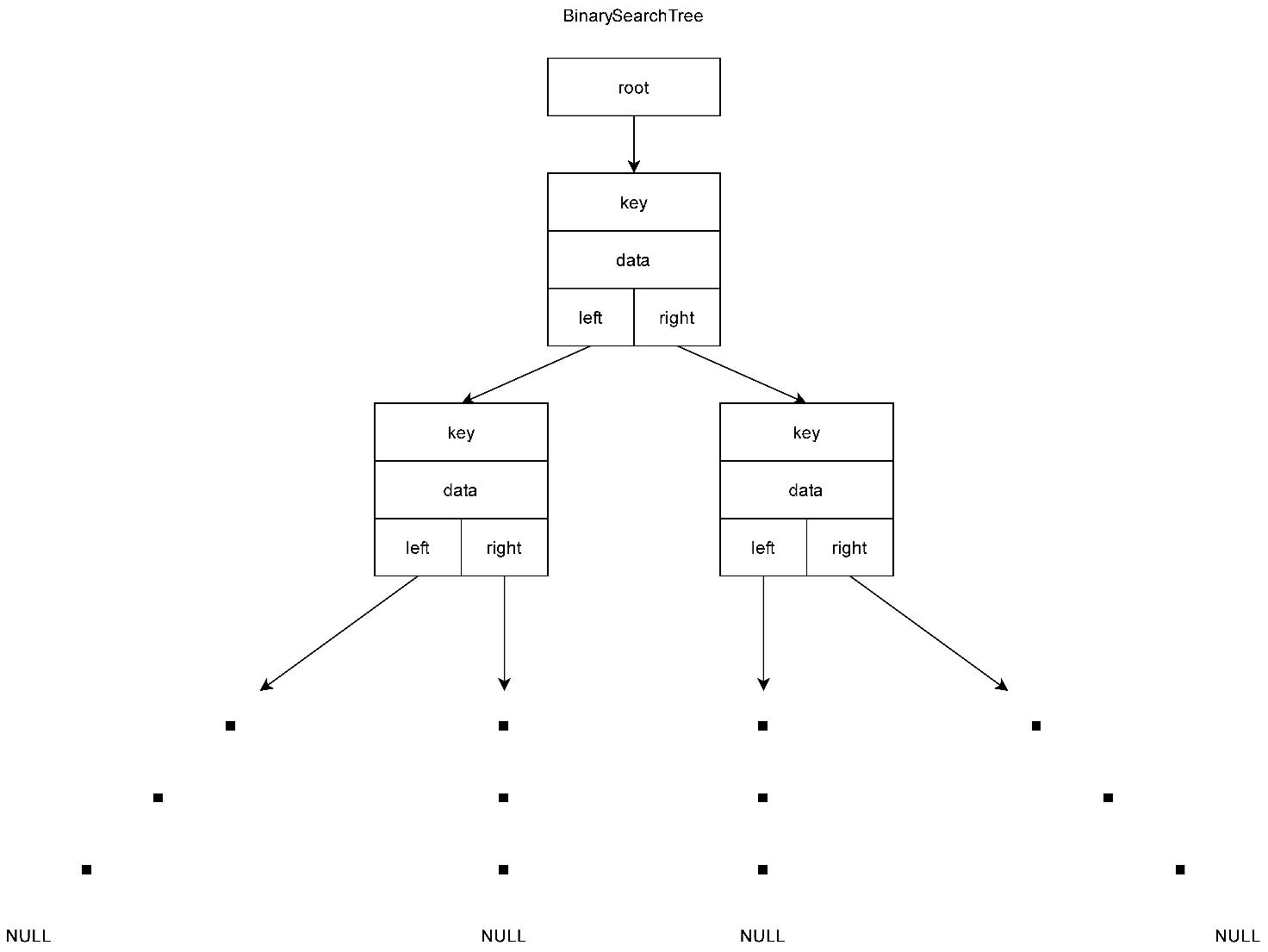
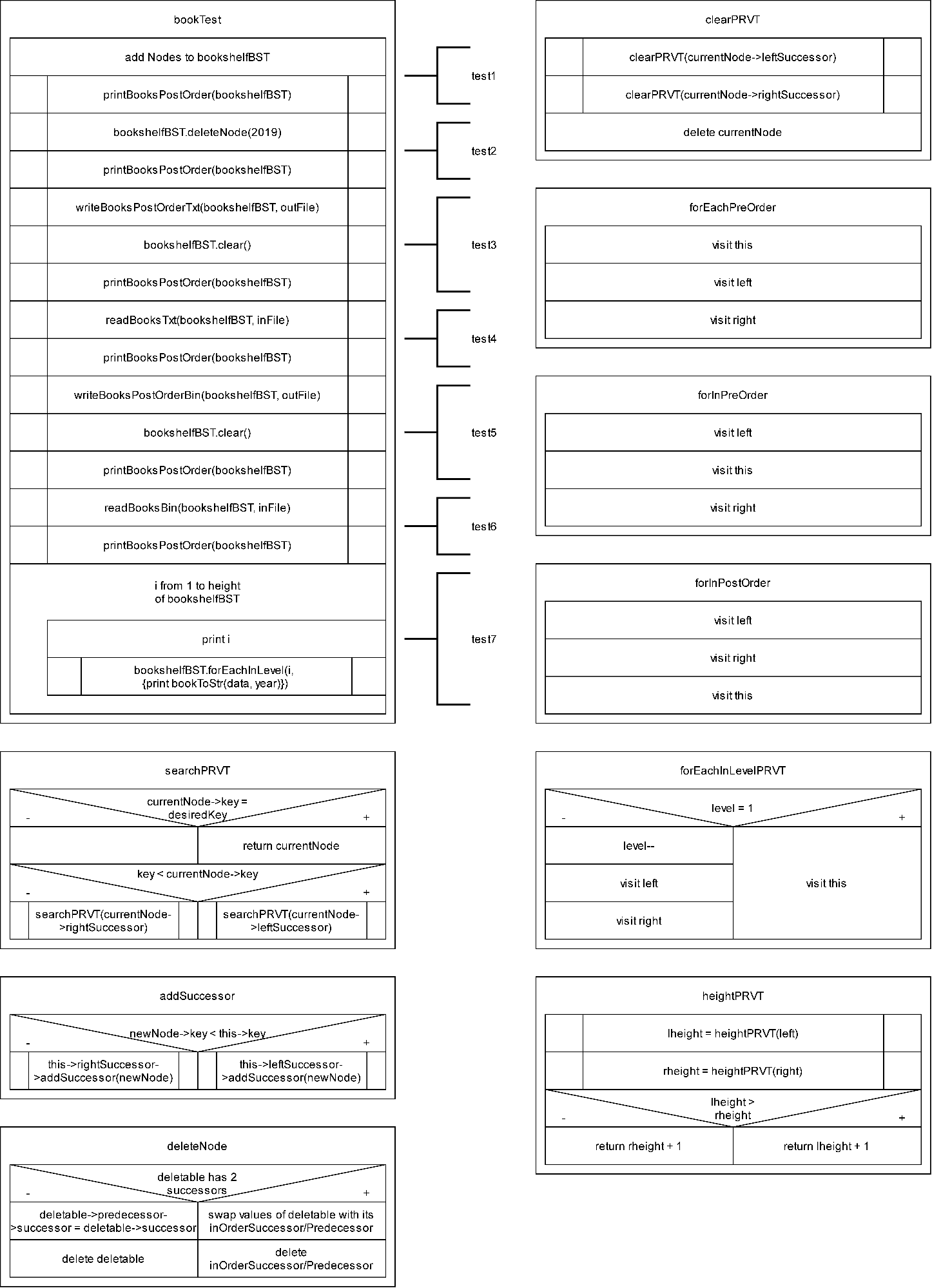


Рисунок 1

# **4. Алгоритми для заданих функцій у вигляді діаграм Нассі-Шнейдермана.**



# **5. Набори тестів для перевірки алгоритмів для заданих функцій (для кожної функції окремий набір тестів).**

Усі тести виконуються послідовно, а отже дії попередніх тестів впливають на результати наступних. Можна вважати, що дії кожного тесту, також, включають дії всіх попередніх, але в запису будуть опущені для стислості. Теж саме стосується результатів.

|  |  |  |  |
| --- | --- | --- | --- |
| № | Назва | Запрограмовані дії | Очікуваний результат |
| 1 | Перевірка додавання вершини | Додавання вершин, виведення усіх вершин у постфіксному порядку. | Виведення усіх вершин у постфіксному порядку. |
| 2 | Перевірка видалення вершини за ключем | Видалення деякої вершини, виведення усіх вершин у постфіксному порядку. | Виведення усіх вершин окрім видаленої у постфіксному порядку. |
| 3 | Перевірка запису у текстовий файл та видалення | Запис усіх вершин у постфіксному порядку у текстовий файл, видалення усіх вершин виведення усіх вершин. | - |
| 4 | Перевірка читання з текстового файлу | Читання з текстового файлу, виведення. | Виведення усіх вершин у постфіксному порядку. |
| 5 | Перевірка запису у бінарний файл та видалення | Запис усіх вершин у постфіксному порядку у бінарний файл, видалення усіх вершин виведення усіх вершин. | - |
| 6 | Перевірка читання з бінарного файлу | Читання з бінарного файлу, виведення. | Виведення усіх вершин у постфіксному порядку. |
| 7 | Перевірка обходу по рівням | Виведення вершин на кожному рівні | Виведення усіх вершин по рівням. |

# **6. Текст програми.**

“main.cpp”

#include "binarySearchTree.h"

#include <vector>

#include<iostream>

#include "solve.h"

void test() {//test with int

binarySearchTree<int, int> bst;

/\*while (true) {//testing destructor

binarySearchTree<int, int> bst1;

std::vector<int> test{ 8, 10, 14, 2, 6, 1, 13, 7, 4, 15, 5, 3 };//{23, 18, 27, 23, 16, 47, 46, 11, 18, 14};

for (auto i : test)

bst1.addNode(i, i);

}\*/

std::vector<int> test{ 8, 10, 14, 2, 6, 1, 13, 7, 4, 15, 5, 3 };//{23, 18, 27, 23, 16, 47, 46, 11, 18, 14};

for (auto i : test)

bst.addNode(i, i);

bst.forEach(preOrder, [](int& data, int key) {

std::cout << key << " ";

});

std::cout << std::endl;

bst.forEach(inOrder, [](int& data, int key) {

std::cout << key << " ";

});

std::cout << std::endl;

bst.forEach(postOrder, [](int& data, int key) {

std::cout << key << " ";

});

std::cout << std::endl;

std::cout << std::endl;

int height = bst.height();

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

bst.forEachInLevel(i, [](int& data, int key) {

std::cout << key << " ";

});

std::cout << std::endl;

}

std::cout << std::endl;

std::cout << \*(bst.search(13)) << std::endl;

bst.deleteNode(8);

std::cout << std::endl;

height = bst.height();

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

bst.forEachInLevel(i, [](int& data, int key) {

std::cout << key << " ";

});

std::cout << std::endl;

}

std::cout << std::endl;

bst.forEach(preOrder, [](int& data, int key) {

std::cout << key << " ";

});

std::cout << std::endl;

bst.forEach(inOrder, [](int& data, int key) {

std::cout << key << " ";

});

std::cout << std::endl;

bst.forEach(postOrder, [](int& data, int key) {

std::cout << key << " ";

});

std::cout << std::endl;

//std::cout << \*(bst.search(2)) << std::endl;

}

int main() {

bookTest();

//test();

return 0;

}

“solve.h”

#ifndef \_\_SOLVE\_H\_\_

#define \_\_SOLVE\_H\_\_

//tests of binary search tree using example where key is int(year), data - book

void bookTest();

#endif

“solve.cpp”

#include "solve.h"

#include "binarySearchTree.h"

#include "books.h"

#include <string>

#include <vector>

#include <utility>//pair

void bookTest() {

std::cout << "Test1: adding nodes and printing them:" << std::endl;

std::vector<std::pair<int, bookData>> bookshelfVec{

{1988, {"Stephen Hawking", "A Brief History of Time"}},

{2001, {"Stephen Hawking", "The Universe in a Nutshell"}},

{1993, {"Stephen Hawking", "Black Holes and Baby Universes"}},

{1892, {"Arthur Conan Doyle", "The Adventures of Sherlock Holmes"}},

{2019, {"Daniel Teslenko", "Against all odds. The way to success"}},

{1922, {"James Joyce", "Ulysses"}},

};

binarySearchTree<int, bookData> bookshelfBST;//int year

for (auto book : bookshelfVec) {

bookshelfBST.addNode(book.second, book.first);

}

printBooksPostOrder(bookshelfBST);

std::cout << std::endl;

std::cout << "Test2: deleting a node(2019) and printing again:" << std::endl;

bookshelfBST.deleteNode(2019);//teslenko

printBooksPostOrder(bookshelfBST);

std::cout << std::endl;

std::cout << "Test3: writing to txt file, deleting whole tree, then trying to print:" << std::endl;

//write to txt file (ofstream << toString())

std::ofstream outFile;

outFile.open("books.txt", std::ios::trunc);

writeBooksPostOrderTxt(bookshelfBST, outFile);

outFile.close();

bookshelfBST.clear();

printBooksPostOrder(bookshelfBST);

std::cout << "Test4: reading from txt file, printing:" << std::endl;

//read from txt file (ifstream >> addNode())

std::ifstream inFile;

inFile.open("books.txt");

readBooksTxt(bookshelfBST, inFile);

inFile.close();

printBooksPostOrder(bookshelfBST);

std::cout << std::endl;

std::cout << "Test5: writing to bin file, deleting whole tree, then trying to print:" << std::endl;

//write to bin file (ofstream << toString())

outFile.open("books.txt", std::ios::binary | std::ios::trunc);

writeBooksPostOrderBin(bookshelfBST, outFile);

outFile.close();

bookshelfBST.clear();

printBooksPostOrder(bookshelfBST);

std::cout << "Test6: reading from bin file, printing:" << std::endl;

//read from bin file (ifstream >> addNode())

inFile.open("books.txt", std::ios::binary);

readBooksBin(bookshelfBST, inFile);

inFile.close();

printBooksPostOrder(bookshelfBST);

std::cout << std::endl;

std::cout << "Test7: level traversal(printing each level):" << std::endl;

int height = bookshelfBST.height();

for (int i = 1; i <= height; i++) {

std::cout << "level " << i << std::endl;

bookshelfBST.forEachInLevel(i, [](bookData& data, int year) {

std::cout << bookToStr(data, year) << std::endl;

});

}

}

“books.h”

#ifndef \_\_BOOKS\_H\_\_

#define \_\_BOOKS\_H\_\_

//header implementing book structure needed for testing of binary search tree class

#include "binarySearchTree.h"

#include <sstream>

#include <string>

#include <fstream>

#include <iostream>

struct bookData {

std::string author;

std::string name;

};

std::string bookToStr(bookData& data, int& year) {

return data.name + " by " + data.author + " in " + std::to\_string(year);

}

//print string of books (bookName by bookAuthor in releaseYear/n)

void printBooksPostOrder(binarySearchTree<int, bookData>& bookshelf) {

int enumerator = 1;

bookshelf.forEach(postOrder, [&](bookData& data, int year) {

std::cout << enumerator << ". " << bookToStr(data, year) << std::endl;

enumerator++;

});

}

//writes books: (name\0author\0year\0\n)//ofstream trunc

void writeBooksPostOrderTxt(binarySearchTree<int, bookData>& bookshelf, std::ofstream& file) {

bookshelf.forEach(postOrder, [&](bookData& data, int year) {

file << data.name << '\0' << data.author << '\0' << year << '\0' << '\n';

});

}

//writes string to binary file formatted like: length of string, string

void putString(std::string& str,std::ofstream& file) {

int size = str.size();

file.write(reinterpret\_cast<char\*>(&size), sizeof(size));

file.write(str.c\_str(), size);

}

//writes books: (sizeof bookName, bookName, sizeof author, author, year)//ofstream trunc bin

void writeBooksPostOrderBin(binarySearchTree<int, bookData>& bookshelf, std::ofstream& file) {

bookshelf.forEach(postOrder, [&](bookData& data, int year) {

putString(data.name, file);

putString(data.author, file);

file.write(reinterpret\_cast<char\*>(&year), sizeof(year));

});

}

void readTxtUntilNULL(std::string& buffer, std::ifstream& file) {

buffer = "";

char currentChar;

file >> std::noskipws >> currentChar;//by default whitespaces(ws) considered as invalid char's

while (currentChar != EOF && currentChar != '\0') {

buffer += currentChar;

file >> currentChar;//serves as file peek too

}

}

//reads books formattedlike that: name\0author\0year\0\n //ifstream

void readBooksTxt(binarySearchTree<int, bookData>& bookshelf, std::ifstream& file) {

bookData book;

int year;

std::string yearStr;

while (file.peek() != EOF && !file.eof()) {

readTxtUntilNULL(book.name, file);

readTxtUntilNULL(book.author, file);

readTxtUntilNULL(yearStr, file);

year = std::stoi(yearStr);//unsecure(exceptions)

file.get();//read one more char \n or EOF

bookshelf.addNode(book, year);

}

}

//reads string from binary file formatted like: length of string, string

std::string getString(std::ifstream& file) {

int size;

file.read(reinterpret\_cast<char\*>(&size), sizeof(size));

char\* buffer = new char[size];

file.read(buffer, size);

std::string ret = "";

ret.append(buffer, size);

delete[]buffer;

return ret;

}

//reads books formattedlike that: sizeof bookName, bookName, sizeof author, author, year //ifstream bin

void readBooksBin(binarySearchTree<int, bookData>& bookshelf, std::ifstream& file) {

bookData book;

int year;

while (file.peek() != EOF && !file.eof()) {

book.name = getString(file);

book.author = getString(file);

file.read(reinterpret\_cast<char\*>(&year), sizeof(year));

bookshelf.addNode(book, year);

}

}

#endif

“binarySearchTree.h”

#ifndef \_\_BINARYSEARCHTREE\_H\_\_

#define \_\_BINARYSEARCHTREE\_H\_\_

//implementation of templated class binary search tree

#include <vector>

#include <functional>

enum traversalType { preOrder, inOrder, postOrder };

template <class KEY, class DATA>

class binarySearchTree {

private:

struct node {

KEY key;

node\* leftSuccessor;

node\* rightSuccessor;

DATA data;

//automatically choose left or right

void addSuccessor(node\* newNode) {

if (newNode->key < this->key)

if (this->leftSuccessor)

this->leftSuccessor->addSuccessor(newNode);

else

this->leftSuccessor = newNode;

else

if (this->rightSuccessor)

this->rightSuccessor->addSuccessor(newNode);

else

this->rightSuccessor = newNode;

}

node(const KEY& key, const DATA& data) {

leftSuccessor = rightSuccessor = NULL;

this->key = key;

this->data = data;

}

};

node\* root;

public:

// construct/destruct:

binarySearchTree() {

root = NULL;

}

~binarySearchTree() {

clear();

}

// public methods:

DATA\* search(const KEY& key) {//Depth first

node\* tmp;

if (this->root)

return &(searchPRVT(key, this->root, tmp)->data);

else

return NULL;

}

//std::function used instead of raw pointer to be able to pass capturing lambdas

//uses func(DATA&, KEY) on each node in traversal order(pre-, post-, in- fix)

void forEach(traversalType traversal, std::function<void(DATA&, KEY)> func) {

switch (traversal) {

case preOrder:

forEachPreOrder(func, this->root);

break;

case inOrder:

forEachInOrder(func, this->root);

break;

case postOrder:

forEachPostOrder(func, this->root);

break;

}

}

//uses func(DATA&, KEY) on each node in level left to right

void forEachInLevel(int level, std::function<void(DATA&, KEY)> func) {

forEachInLevelPRVT(level, func, this->root);

}

int height() {

return heightPRVT(this->root);

}

//deletes all nodes

void clear() {

clearPRVT(root);

root = NULL;

}

void addNode(const DATA& data, const KEY& key) {

node\* newNode = new node(key, data);

if (this->root)

this->root->addSuccessor(newNode);

else

this->root = newNode;

}

//find and delete node by key

void deleteNode(const KEY& value) {

node\* predecessor = root, \* newNode;

node\* deletable = searchPRVT(value, this->root, predecessor);

if (!deletable)

return;//pass

else if (deletable->rightSuccessor && deletable->leftSuccessor) {

int predecessorHeight = 0;

node\* inOrderPredecessor = deletable->rightSuccessor;

while (inOrderPredecessor->leftSuccessor) {

inOrderPredecessor = inOrderPredecessor->leftSuccessor;

predecessorHeight++;

}

int successorHeight = 0;

node\* inOrderSuccessor = deletable->leftSuccessor;

while (inOrderSuccessor->rightSuccessor) {

inOrderSuccessor = inOrderSuccessor->rightSuccessor;

successorHeight++;

}

newNode = (successorHeight > predecessorHeight) ? inOrderSuccessor : inOrderPredecessor;//take the one from lowest level

searchPRVT(newNode->key, this->root, predecessor);//predecessor of newNode

deletable->key = newNode->key;//replace key&value in deletable with ones from newNode

deletable->data = newNode->data;

deletable = newNode;//to delete out of if

newNode = NULL;

}

else//one or no sucessors

newNode = !deletable->rightSuccessor ? deletable->leftSuccessor : deletable->rightSuccessor;//choosing single available successor or NULL

disconnect(predecessor, deletable, newNode);

delete deletable;

}

private:

//replaces succesor of predecessor with newSuccessor

void disconnect(node\* predecessor, node\* successor, node\* newSuccessor) {

if (predecessor->leftSuccessor == successor)

predecessor->leftSuccessor = newSuccessor;

else if (predecessor->rightSuccessor == successor)

predecessor->rightSuccessor = newSuccessor;

}

void clearPRVT(node\* currentNode) {

if (!currentNode)

return;

if (currentNode->leftSuccessor)

clearPRVT(currentNode->leftSuccessor);

if (currentNode->rightSuccessor)

clearPRVT(currentNode->rightSuccessor);

delete currentNode;

}

node\* searchPRVT(const KEY& key, node\* currentNode, node\*& predcessor) {

if (currentNode->key == key)

return currentNode;

else if (key < currentNode->key)

if (currentNode->leftSuccessor) {

predcessor = currentNode;

return searchPRVT(key, currentNode->leftSuccessor, predcessor);

}

else

return NULL;

else //if (key > currentNode->key)

if (currentNode->rightSuccessor) {

predcessor = currentNode;

return searchPRVT(key, currentNode->rightSuccessor, predcessor);

}

else

return NULL;

}

void forEachPreOrder(std::function<void(DATA&, KEY)> func, node\* currentNode) {

if (!currentNode)

return;

func(currentNode->data, currentNode->key);

if (currentNode->leftSuccessor)

forEachPreOrder(func, currentNode->leftSuccessor);

if (currentNode->rightSuccessor)

forEachPreOrder(func, currentNode->rightSuccessor);

}

void forEachInOrder(std::function<void(DATA&, KEY)> func, node\* currentNode) {

if (!currentNode)

return;

if (currentNode->leftSuccessor)

forEachInOrder(func, currentNode->leftSuccessor);

func(currentNode->data, currentNode->key);

if (currentNode->rightSuccessor)

forEachInOrder(func, currentNode->rightSuccessor);

}

void forEachPostOrder(std::function<void(DATA&, KEY)> func, node\* currentNode) {

if (!currentNode)

return;

if (currentNode->leftSuccessor)

forEachPostOrder(func, currentNode->leftSuccessor);

if (currentNode->rightSuccessor)

forEachPostOrder(func, currentNode->rightSuccessor);

func(currentNode->data, currentNode->key);

}

void forEachInLevelPRVT(int level, std::function<void(DATA&, KEY)> func, node\* currentNode) {

if (!currentNode || level < 1)

return;

else if (level == 1)

func(currentNode->data, currentNode->key);

else//level > 1

if (currentNode->leftSuccessor)

forEachInLevelPRVT(level - 1, func, currentNode->leftSuccessor);

if (currentNode->rightSuccessor)

forEachInLevelPRVT(level - 1, func, currentNode->rightSuccessor);

}

int heightPRVT(node\* currentNode) {

if (!currentNode)

return 0;

else {

int lheight = heightPRVT(currentNode->leftSuccessor);

int rheight = heightPRVT(currentNode->rightSuccessor);

return (lheight > rheight) ? (lheight + 1) : (rheight + 1);

}

}

};

#endif

# **7. Результати тестування програми та їх аналіз.**



Рисунок 2

Результати відповідають очікуванням, програма працює коректно.

# **8. Висновки щодо структури бінарного дерева.**

Бінарне дерево пошуку – бінарне дерево у якому кожна вершина має значення ключа більше ніж лівий нащадок, і менше ніж правий. Додавання, видалення та пошук вершин у дереві швидший ніж у звичайному лінійному списку. Але ця структура рекурсивна, через це з’являються дві можливі проблеми:

1. Обмежений розмір контейнеру – через глибину рекурсії буде переповнюватися стек виклику.
2. Програмний стек – його потрібно використовувати щоб перейти від рекурсивних функцій до ітеративних.

Також кожна вершина має мати два вказівники на нащадків, у той самий час однозв’зному лінійному списку достатнього одного.

Якщо підсумувати можна сказати, що отримуються переваги прийнятною ціною, через це дана структура має свої області використання.