МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

«Харківський Політехнічний Інститут»

Кафедра «Стратегічне управління»

Звіт з лабораторної роботи №3

«Бінарні дерева та червоно-чорні дерева»

з дисципліни

"Алгоритми та структури даних"

Варіант № 6

Перевірив:

ст. викл. каф. стратегічного

управління Мошко Є.О.

Виконав: Колій Дмитро

ст. гр. КН-320б

Харків – 2022

Мета: Набути практичних вмінь та навичок опрацювання динамічних структур даних, представлених у вигляді бінарних та червоно-чорних дерев.

Індивідуальне завдання:

1. Написати програму, в якій дані варіанту структури записуються в бінарне дерево.

2. Знайти середнє значення одного з числових полів, зчитуючи дані з дерева.

3. Надрукувати значення бінарного дерева:

а) при прямому обході дерева;

б) при зворотному обході дерева;

в) при симетричному обході дерева.

4. Дописати функцію видалення з пам’яті всього бінарного дерева.

5. Створити рекурсивну функцію, яка:

− видаляє ліве піддерево, і ліву гілку занулює;

− видаляє праве піддерево, і праву гілку занулює;

− видаляє сам вузол, потім зануливши вказівник на нього. Наприкінці програми видалити з пам’яті дерево.

6. "Пересипати" дані з першого дерева у друге дерево того ж типу, тільки організованого за першим числовим ключем та роздрукувати його (а перше дерево стерти).

7. Розфарбувати вершини заданого бінарного дерева в червоний і чорний кольори так, щоб воно стало червоно-чорним деревом.

8. Провести перевірку властивості червоно-чорного дерева, а саме:

− кожен вузол промаркований червоним або чорним кольором;

− корінь дерева – чорний;

− кінцеві вузли дерева – чорні;

− біля червоного вузла батьківський вузол — чорний;

− усі прості шляхи з будь-якого вузла до листя містять однакову кількість чорних вузлів.

9. У червоно-чорному дереві виконати додавання і видаленні вузлів та провести перевірку властивості червоно-чорного дерева.

Наприкінці програми видалити з пам’яті дерево.

Варіант завдання вибирається за номером студента в списку групи:

|  |  |
| --- | --- |
| № варіанту | Дані |
| 6 | 1. Прізвище працівника компанії  2. Особистий номер працівника  3. Оклад працівника |

Схема алгоритму виконання функцій

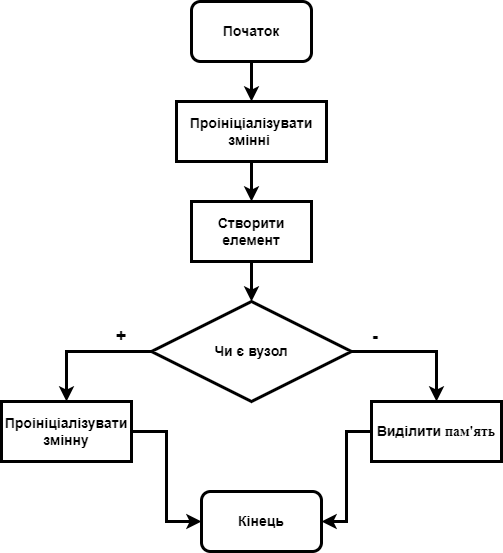
****

Рисунок 1 – Блок-схема для функції створення бінарного дерева

Програмна реалізація створення бінарного дерева :

void creata\_tree(Tlist\*& root) {

int col\_elem(0), way\_zapoln;

info per\_s\_data;

vvedinie\_data(col\_elem, 0, INT\_MAX, "Введите количество елементов дерева :");

vvedinie\_data(way\_zapoln, 0, 1, "Как вы хотите заполнить все узлы? 1 - вручную; 0 - из файла.");

for (int i = 0; i < col\_elem; i++)

{

if (way\_zapoln == 1) {

cout << "Ввод значений для узла №" << i << endl;

vvedenia\_data\_stroka(per\_s\_data.sec\_name\_emp, "Введите фамили :");

vvedinie\_data(per\_s\_data.num\_emp, 1000, 9999, "Введите номер паспорта :");

vvedinie\_data(per\_s\_data.oklad\_emp, 0, 150, "Введите возраст :");

}

else {

inizial\_var(per\_s\_data);

}

add\_node(per\_s\_data, root, true);

}

}

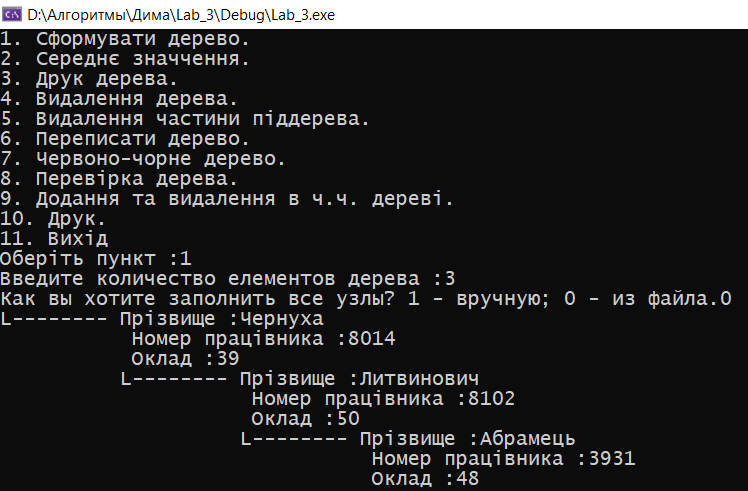


Рисунок 2 – Демонстрація роботи функції створення стеку



Рисунок 3 – Алгоритм виконання розрахунку середнього арифметичного

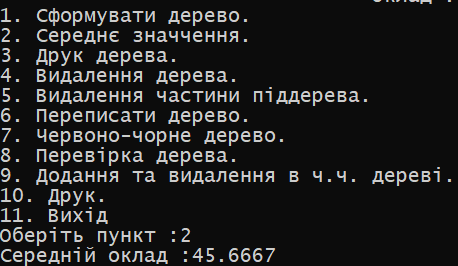


Рисунок 4 – Результат виконання функці

class Average\_item {// сред знач

public:

int sum = 0;

int count = 0;

void getAverage(Tlist\* node) {

if (node == nullptr) {

return;

}

sum += node->data.oklad\_emp;

count++;

getAverage(node->left);

getAverage(node->right);

}

};



Рисунок 5  – Алгоритм виконання функції прямого обходу

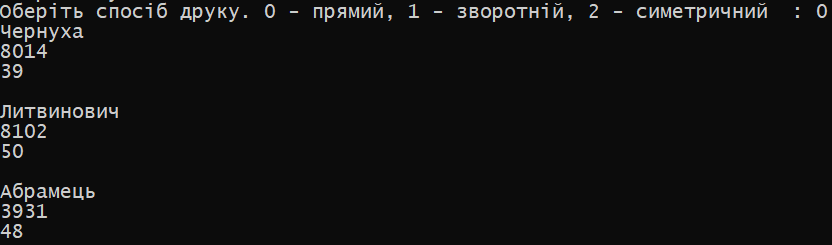


Рисунок 6 – Результат прямого обходу

void PreOrder\_BinaryTree(Tlist\* Node) {

if (Node != NULL) {

cout << Node->data.sec\_name\_emp << endl;

cout << Node->data.num\_emp << endl;

cout << Node->data.oklad\_emp << endl << endl;

PreOrder\_BinaryTree(Node->left);

PreOrder\_BinaryTree(Node->right);

}

}

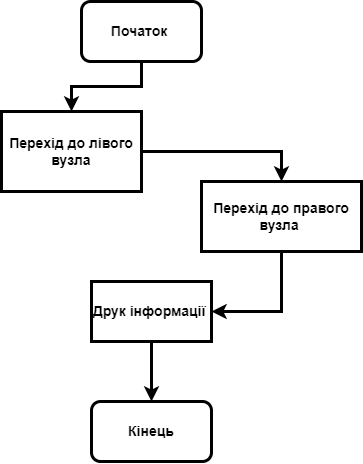


Рисунок 7 – Алгоритм виконання функції зворотнього обходу

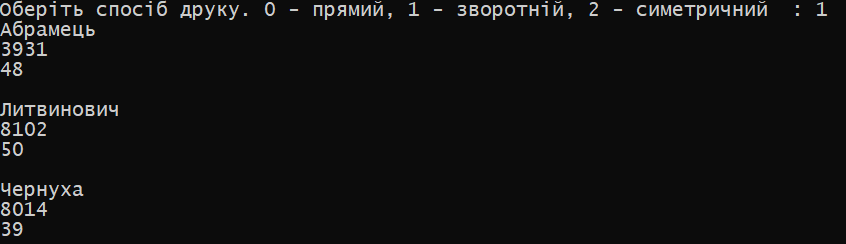


Рисунок 8 – результат зворотнього обходу

void PostOrder\_BinaryTree(Tlist\* Node) {

if (Node != NULL) {

PostOrder\_BinaryTree(Node->left);

PostOrder\_BinaryTree(Node->right);

cout << Node->data.sec\_name\_emp << endl;

cout << Node->data.num\_emp << endl;

cout << Node->data.oklad\_emp << endl << endl;

}

}

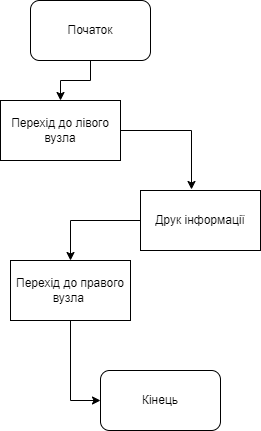


Рисунок 9 – Алгоритм виконання функції симетричного обходу

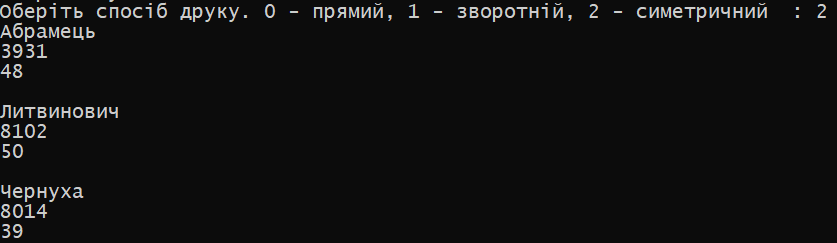


Рисунок 10 – Результат виконання симетричного обходу

void Symmetricorder\_BinaryTree(Tlist\* Node) {

if (Node != NULL) {

PostOrder\_BinaryTree(Node->left);

cout << Node->data.sec\_name\_emp << endl;

cout << Node->data.num\_emp << endl;

cout << Node->data.oklad\_emp << endl << endl;

PostOrder\_BinaryTree(Node->right);

}

}

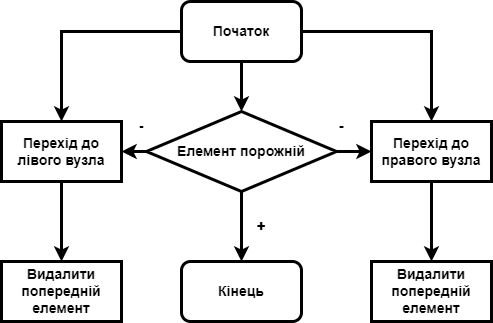


Рисунок 11 – Алгоритм виконання функції видалення дерева

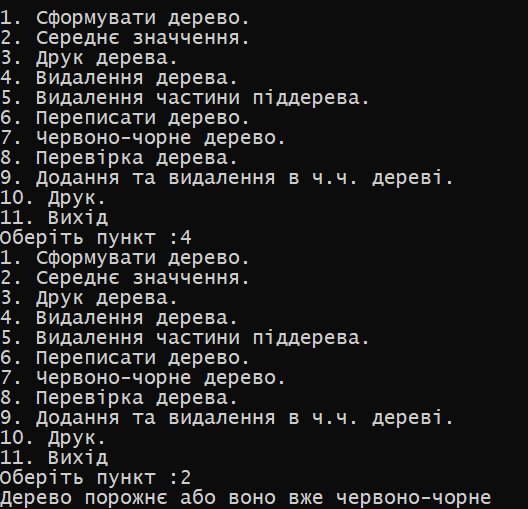


Рисунок 12 – Результат видалення дерева

void del(Tlist\*& tree) {

if (tree->left) del(tree->left);

if (tree->right) del(tree->right);

delete tree;

tree = nullptr;

}

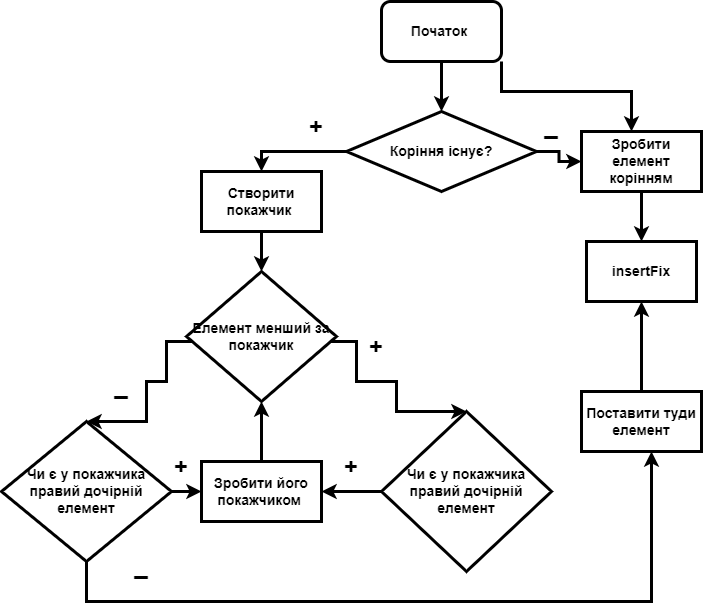


Рисунок 13 – Алгоритм виконання функції додавання елементу до ЧЧ дерева

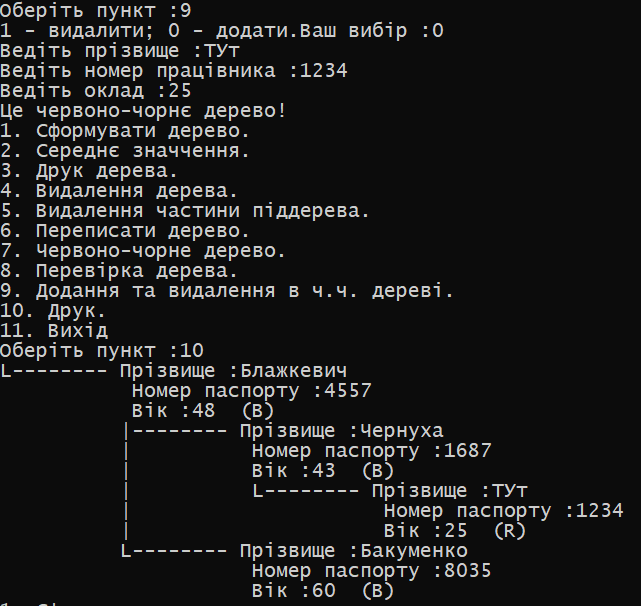


Рисунок 14 – Результат додавання елементу до ЧЧ дерева

void insert(info key) {

NodePtr node = new Node;

node->parent = nullptr;

node->data = key;

node->left = TNULL;

node->right = TNULL;

node->color = RED;

NodePtr y = nullptr;

NodePtr x = this->root;

while (x != TNULL) {

y = x;

if (node->data.sec\_name\_emp <= x->data.sec\_name\_emp) {

x = x->left;

}

else {

x = x->right;

}

}

node->parent = y;

if (y == nullptr) {

root = node;

}

else if (node->data.sec\_name\_emp < y->data.sec\_name\_emp) {

y->left = node;

}

else {

y->right = node;

}

if (node->parent == nullptr) {

node->color = BLACK;

return;

}

if (node->parent->parent == nullptr) {

return;

}

insertFix(node);

}

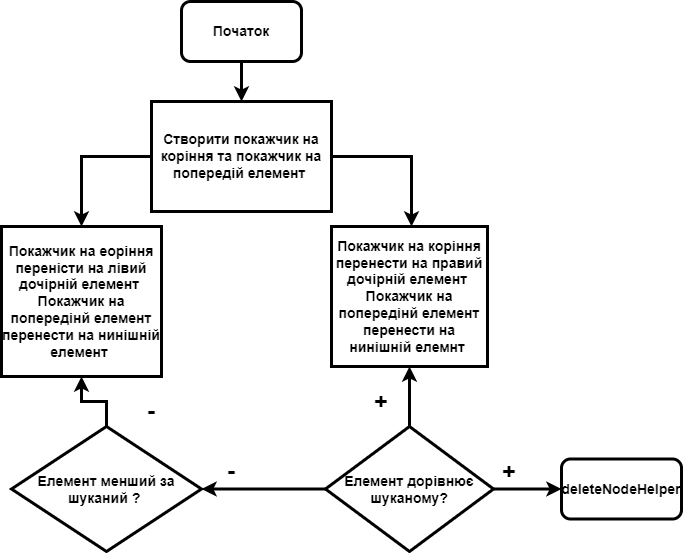


Рисунок 15 – Алгоритм видалення елементу з ЧЧ дерева

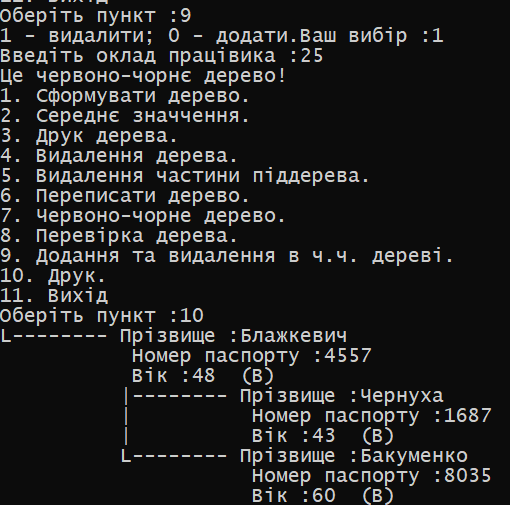


Рисунок 16 – Результат видалення елементу з ЧЧ дерева

void deleteNodeHelper(NodePtr node, info key) {

NodePtr z = TNULL;

NodePtr x, y;

while (node != TNULL) {

if (node->data.oklad\_emp == key.oklad\_emp) {

z = node;

}

if (node->data.sec\_name\_emp <= key.sec\_name\_emp) { //МОЖЕТ НЕ ВЛЕВО А В ПРАВО

node = node->right;

}

else {

node = node->left;

}

}

if (z == TNULL) {

cout << "Key not found in the tree" << endl;

return;

}

y = z;

int y\_original\_color = y->color;

if (z->left == TNULL) {

x = z->right;

rbTransplant(z, z->right);

}

else if (z->right == TNULL) {

x = z->left;

rbTransplant(z, z->left);

}

else {

y = minimum(z->right);

y\_original\_color = y->color;

x = y->right;

if (y->parent == z) {

x->parent = y;

}

else {

rbTransplant(y, y->right);

y->right = z->right;

y->right->parent = y;

}

rbTransplant(z, y);

y->left = z->left;

y->left->parent = y;

y->color = z->color;

}

delete z;

if (y\_original\_color == 0) {

deleteFix(x);

}

}

Висновок: Під час виконання даної лабораторної роботи були отриманні навички роботи з бінарним та червоно-чорним деревом. Я ознайомитися із основними способами організації бінарних дерев, перетворення бінарного дерева в червоно-чорне та особливостями їх програмної реалізації. Набув практичних вмінь опрацювання динамічних структур даних, представлених у вигляді бінарних та червоно-чорних дерев.

Бінарне дерево складається з щонайбільше двох вузлів, які часто називають лівим і правим вузлами, і елементом даних. Найбільш верхній вузол дерева називається кореневим вузлом, а лівий і правий вказівники направляють на менші підряди з обох сторін.

Червоно-чорне дерево — різновид самозбалансованого бінарного дерева пошуку, вершини якого мають додаткові властивості, зокрема «колір» (червоний або чорний). Ці біти кольору використовуються для забезпечення того, щоб дерево залишалося приблизно збалансованим при виконанні операцій вставки та видалення.

Обмеження при будуванні червоно-чорного дерева:

* кожен вузол має колір червоний або чорний;
* корінь дерева – чорний;
* листя дерева – чорні;
* у червоного вузла батьківський вузол – чорний;
* всі прості шляхи з будь-якого вузла до листя має однакові кількість чорних вузлів.

За допомогою цих обмежень дерево є самобалансуючим, що гарантує логарифмічний ріст висоти дерева від кількості вузлів. Це дозволяє швидко виповнювати основні операції з деревом: додання, видалення та пошук вузла.