МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

“ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ”

КАФЕДРА ПРОГРАМНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ УПРАВЛІННЯ

ЗВІТ

З ЛАБОРАТОРНОЇ РОБОТИ №3

ЗА ТЕМОЮ “БАЗОВІ СТРУКТУРИ ДАНИХ: ЧЕРВОНОЧОРНІ ДЕРЕВА”

Виконав студент

групи КН-221д

Кукуєв Руслан Олександрович

Перевірив

Солонська С.В.

Харків 2021

ЛАБОРАТОРНА РОБОТА №3

Тема: БАЗОВІ СТРУКТУРИ ДАНИХ: ЧЕРВОНОЧОРНІ ДЕРЕВА.

Мета роботи: познайомитися з червоно-чорними деревами та отримати навички програмування алгоритмів, що їх обробляють.

**Завдання:**

Розробити програму, яка читає з клавіатури числа N, M (1 < N, M < 256); послідовність N ключів (цілих, дійсних чисел або рядків (до 255 символів) в залежності від варіанту завдання); послідовність M ключів. Програма зберігає першу послідовність до червоно-чорного дерева.

Кожного разу, коли до дерева додається новий елемент, потрібно вивести статистику (згідно варіанту завдання).

1.Мінімальний елемент та його колір;

2.Максимальний елемент та його колір.

Після побудови дерева для кожного елементу x другої послідовності потрібно вивести результати наступних операцій над деревом (згідно варіанту завдання).

1 Чи є елемент у дереві та його колір.

2 Successor(x) та його колір.

3 Predecessor(x) та його колір.

Використовувати готові реалізації структур даних (наприклад, STL) заборонено, але можна використати реалізацію рядків (наприклад, std::string у C++).

3.3 Варіанти завдань

1 1.1.

2 1.2.

3 1.3.

4 2.1.

5 2.2.

6 2.3.

**Варіант №6**

#include <iostream>

using namespace std;

enum NodeColor

{

BLACK = 0,

RED = 1

};

struct Node

{

Node(int data)

{

this->data = data;

}

int data;

NodeColor color; /\* node color (BLACK, RED) \*/

struct Node \*left = NULL; /\* left child \*/

struct Node \*right = NULL; /\* right child \*/

struct Node \*parent = NULL; /\* parent \*/

};

NodeColor getColor(Node \*node)

{

if (node == NULL)

return BLACK;

return node->color;

}

void setColor(Node \*node, NodeColor color)

{

if (node == NULL)

return;

node->color = color;

}

void insertNode(Node \*root, Node \*ptr)

{

if (root->data > ptr->data)

{

if (root->left == NULL)

{

root->left = ptr;

ptr->parent = root;

}

else

{

insertNode(root->left, ptr);

}

}

else if (root->data < ptr->data)

{

if (root->right == NULL)

{

root->right = ptr;

ptr->parent = root;

}

else

{

insertNode(root->right, ptr);

}

}

}

void rotateLeft(Node \*ptr)

{

Node \*right\_child = ptr->right;

ptr->right = right\_child->left;

if (ptr->right != NULL)

ptr->right->parent = ptr;

right\_child->parent = ptr->parent;

if (ptr->parent != NULL && ptr == ptr->parent->left)

ptr->parent->left = right\_child;

else

ptr->parent->right = right\_child;

right\_child->left = ptr;

ptr->parent = right\_child;

}

void rotateRight(Node \*ptr)

{

Node \*left\_child = ptr->left;

ptr->left = left\_child->right;

if (ptr->left != NULL)

ptr->left->parent = ptr;

left\_child->parent = ptr->parent;

if (ptr->parent != NULL && ptr == ptr->parent->left)

ptr->parent->left = left\_child;

else

ptr->parent->right = left\_child;

left\_child->right = ptr;

ptr->parent = left\_child;

}

void fixInsertNode(Node \*ptr)

{

Node \*parent = NULL;

Node \*grandparent = NULL;

while (getColor(ptr) == RED && getColor(ptr->parent) == RED)

{

parent = ptr->parent;

grandparent = parent->parent;

if (parent == grandparent->left)

{

Node \*uncle = grandparent->right;

if (getColor(uncle) == RED)

{

setColor(uncle, BLACK);

setColor(parent, BLACK);

setColor(grandparent, RED);

ptr = grandparent;

}

else

{

if (ptr == parent->right)

{

rotateLeft(parent);

ptr = parent;

parent = ptr->parent;

}

rotateRight(grandparent);

ptr = parent;

}

}

else

{

Node \*uncle = grandparent->left;

if (getColor(uncle) == RED)

{

setColor(uncle, BLACK);

setColor(parent, BLACK);

setColor(grandparent, RED);

ptr = grandparent;

}

else

{

if (ptr == parent->left)

{

rotateRight(parent);

ptr = parent;

parent = ptr->parent;

}

rotateLeft(grandparent);

ptr = parent;

}

}

}

}

int minValueNode(Node \*node)

{

Node \*ptr = node;

while (ptr->left != NULL)

ptr = ptr->left;

return ptr->data;

}

int maxValueNode(Node \*node)

{

Node \*ptr = node;

while (ptr->right != NULL)

ptr = ptr->right;

return ptr->data;

}

void insertValue(Node \*root, int n)

{

Node \*node = new Node(n);

insertNode(root, node);

fixInsertNode(node);

}

void predecessor(Node \*root, int n)

{

if (root == NULL)

{

cout << "Нету предшественника у ключа: " << n << endl;

return;

}

if (root->data == n)

{

cout << "Цвет предшественника: " << (getColor(root->parent) == BLACK ? "Черный" : "Красный") << endl;

return;

}

if (root->data > n)

{

predecessor(root->left, n);

}

else if (root->data < n)

{

predecessor(root->right, n);

}

}

void Init(short int &N, short int &M, short int &cycleCounter, int &mainNodeData)

{

while (N <= 1 || N >= 256 || M <= 1 || M >= 256)

{

cout << "Сколько значений хотите ввести?" << endl;

cin >> N;

cout << "Сколько значений хотите проверить?" << endl;

cin >> M;

}

cout << "Введите значение начального узла:";

cin >> mainNodeData;

}

int main()

{

short int N, M, cycleCounter;

int mainNodeData;

Init(N, M, cycleCounter, mainNodeData);

Node \*mainNode = new Node(mainNodeData);

while (cycleCounter < (N - 1))

{

int newNodeData;

cout << "Введите значение узла:";

cin >> newNodeData;

insertValue(mainNode, newNodeData);

cout << "Мин:" << minValueNode(mainNode) << "Макс:" << maxValueNode(mainNode) << endl;

cycleCounter++;

}

cycleCounter = 0;

while (cycleCounter < M)

{

int findNodeData;

cout << "Найти предшественника узла:";

cin >> findNodeData;

predecessor(mainNode, findNodeData);

cycleCounter++;

}

}

**Висновок:** виконавши лабораторну роботу №3, я познайомився з червоно-чорними деревами та отримав навички програмування алгоритмів, що їх обробляють.