МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра прикладной математики и информатики

ОТЧЁТ

по лабораторной работе №4 «Рефакторинг лабораторной работы №4»

по дисциплине

Структуры данных

(наименование дисциплины)

РУКОВОДИТЕЛЬ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Санников А.Н.

(подпись) (фамилия, и.,о.)

СТУДЕНТ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Зырянов Е.А.

(подпись) (фамилия, и.,о.)

22-ПМ-1

(шифр группы)

Работа защищена «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_

С оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

***Задание на лабораторную работу:*** Осуществить рефакторинг лабораторной работы №3 «N-ое дерево».

***Задание проверяемой лабораторной работы****:* Дано N-дерево. Найти все поддеревья, листья которых находятся в заданном диапазоне высот от корня поддерева.

|  |  |  |
| --- | --- | --- |
| Ошибка | Было | Стало |
| Отсутствие комментариев | Нет комментариев на сложное выражение | Добавления комментариев |
| Отсутствие пробелов в операторах выражения | В операторах отсутствовали пробелы | Добавлены пробелы |

До ревью:

#include <iostream>

#include<fstream>

#include<string>

using namespace std;

// ===============================================================

// Структуры:

// 1. Дерево (Tree):

// int n - макс. кол-во сыновей,

// Node \*root - корневой узел

// 2. Узел дерева (Node):

// int value - значение, хранимое в узле,

// Node\*\* sons - массив длиной n, содержащий ссылки на сыновей.

// ================================================================

struct Node

{

int value;

Node\*\* sons;

Node(int val, int n) : value(val)

{

sons = new Node \*[n];

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

{

sons[i] = nullptr;

}

}

};

struct Tree

{

int n;

Node\* root;

Tree(int num) : n(num) {}

};

// ===============================================================

// Функции для решения задачи

// ================================================================

// Рекурсивно обходит в глубину поддерево дерева tree с корнем поддерева node

// Проверяет, что все листья поддерева лежат в диапазоне глубины от корня [min, max]

// Возвращает строковое представление поддерева, если все листья находятся в диапазоне

// Возвращает пустую строку, если хотя бы один лист лежит за пределами диапазона

// level - текущий уровень глубины, на котором находится node (корень изначального дерева: level = 1)

string checkSubtree(Tree\* tree, Node\* node, int n, int min, int max, string path, int level) {

if (node == nullptr)

return "";

bool childFree = true;

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

if (node -> sons[i] != nullptr)

childFree = false;

}

path += "\nNode " + to\_string(node -> value) + ": ";

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

path += ((node -> sons[i] == nullptr) ? "None" : to\_string(node -> sons[i] -> value));

if (i != (n - 1))

path += " and ";

}

if (childFree) {

if (level < min || level > max)

return "";

return path;

}

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

if (node -> sons[i] != nullptr)

{

path = checkSubtree(tree, node->sons[i], n, min, max, path, level + 1);

if (path.empty())

return "";

}

}

return path;

}

// Рекурсивно обходит в глубину (сверху вниз, слева направо) все узлы дерева tree с корнем node

// С помощью функции checkSubtree выполняет проверку, что все листья поддерева с корнем в текущем узле

// лежат в диапазоне [min, max]. Если условие выполняется, выводит структуру найденного поддерева в

// консоль.

// level - глубина, на которой находится node относительно корня изначального дерева tree

// (корень изначального дерева: level = 1)

bool findAllSubtreesInRange(Tree\* tree, Node\* node, int n, int min, int max, int level = 1) {

if (node == nullptr) {

return false;

}

bool found = false;

string path = checkSubtree(tree, node, n, min, max, "", level);

if (!path.empty()) {

cout << "////////////////////////////////////////////////////////////////////////////////////////////" << endl;

cout << "Subtree found:" << endl;

cout << path << endl << endl;

found = true;

}

if (level == max)

return found;

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

if (node->sons[i] != nullptr)

if (findAllSubtreesInRange(tree, node -> sons[i], n, min, max, level + 1))

found = true;

}

return found;

}

// Рекурсивная функция для ввода узлов дерева

// n - макс. количество потомков

// displayPrompts - показывать запросы ввода и подсказки в консоли

// (true - для ввода с клавиатуры, false - при чтении файла)

// Для каждого узла:

// Запрашивает ввод целочисленного значения (номера) узла node -> value

// n раз просит сообщить, имеется ли сын в соответствующей позиции в массиве: 1 - да, 0 - нет

// Если сын имеется, рекурсивно переходит к заполнению информации о нём

// Таким образом, выполняет построение дерева в глубину (сначала сверху вниз, затем слева направо, поднимаясь к корню)

// Возвращает Node\*

Node\* inputTree(istream& input, int n, bool displayPrompts = true) {

int rootValue;

if (displayPrompts) {

cout << "Введите значение корня дерева: ";

}

while (!(input >> rootValue)) {

cout << "Ошибка! Введите целое число: ";

input.clear();

input.ignore(32767, '\n');

}

Node\* root = new Node(rootValue, n);

int reply;

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

if (displayPrompts) {

cout << "Есть ли сын №" << (i + 1) << " у узла " << root->value << "? (1 - да, 0 - нет): ";

}

while (!(input >> reply) || (reply != 0 && reply != 1)) {

cout << "Ошибка ввода! Введите 0 или 1: ";

input.clear();

input.ignore(32767, '\n');

}

if(reply) {

root -> sons[i] = inputTree(input, n, displayPrompts);

}

}

return root;

}

// Запрашивает ввод максимального количества сыновей у каждого узла (арность) при создании дерева: tree -> n

// После чего запускает рекурсивную функцию inputTree для ввода узлов дерева, начяиная с корня tree -> root

// Возвращает Tree\* tree

Tree\* inputSize(istream& input, bool displayPrompts = true) {

int n;

if (displayPrompts) {

cout << "Введите значение n - арности дерева: ";

}

// Повторно заправшивает ввод, пока введено не целое число, которое больше нуля

while (!(input >> n) || n <= 0) {

cout << "Ошибка! Введите целое число > 0: ";

input.clear();

input.ignore(32767, '\n');

}

Tree\* tree = new Tree(n);

tree -> root = inputTree(input, n, displayPrompts);

return tree;

}

// Вывод дерева с корнем node и арностью n при обходе в глубину (сверху вниз, слева направо)

void printTreeDFS(Node \*node, int n) {

if (node == nullptr)

return;

string sons;

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

sons += ((node -> sons[i] == nullptr) ? "None" : to\_string(node -> sons[i] -> value));

if (i != (n - 1))

sons += " and ";

}

cout << "Node "

<< node->value << ": " << sons << endl;

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

printTreeDFS(node -> sons[i], n);

}

}

// Вывод дерева при передачи структуры Tree\*, используя printTreeDFS(\*node, n)

void printTree(Tree \*tree) {

printTreeDFS(tree -> root, tree -> n);

}

// Чтобы окно командной строки не закрывалось при запуске .exe

void pause() {

string input;

cout << "\nВведите любой символ и нажмите Enter, чтобы выйти...";

cin >> input;

}

int main() {

setlocale(LC\_ALL, "ru\_RU.UTF-8");

system("chcp 65001");

string filename;

char choice;

bool inputValid = false;

do {

cout << "Выберите источник ввода (k - клавиатура, f - файл): ";

cin >> choice;

Tree\* tree;

if (choice == 'k') {

tree = inputSize(cin);

inputValid = true;

}

else if (choice == 'f') {

while(true) {

cout << "Введите имя файла: ";

cin >> filename;

ifstream inputFile(filename);

if (!inputFile.is\_open()) {

cerr << "Ошибка открытия файла! Повторите..." << endl;

continue;

}

tree = inputSize(inputFile, false);

inputFile.close();

inputValid = true;

break;

}

}

else

{

cout << "Некорректный выбор. Выберите k или f." << endl;

}

if (inputValid) {

cout << "Введённое дерево (DFS):" << endl;

printTree(tree);

int min, max;

cout << "Проограмма выполняет поиск всех поддеревьев, листья которых лежат" << endl

<< "в диапазоне глубины [min, max]. Глубина корня исходного дерева = 1." << endl;

cout << "Введите значение min: ";

// Повторно заправшивает ввод, пока введено не целое число, которое больше нуля

while (!(cin >> min)) {

cerr << "Ошибка! Введите целое число: ";

}

cout << "Введите значение max >= " << min << ": ";

// Повторно заправшивает ввод, пока введено не целое число, которое больше нуля

while (!(cin >> max) || max < min) {

cerr << "Ошибка! Введите целое число >= min (" << min << "): ";

}

if (!findAllSubtreesInRange(tree, tree -> root, tree -> n, min, max))

cout << "Для заданного дерева не было найдено поддеревьев, "

"все листья которых располагались бы в диапазоне [" << min << ", " <<

max << "]" << endl;

}

} while (!inputValid);

pause();

}

После:

#include <iostream>

#include<fstream>

#include<string>

using namespace std;

// ===============================================================

// Структуры:

// 1. Дерево (Tree):

// int n - макс. кол-во сыновей,

// Node \*root - корневой узел

// 2. Узел дерева (Node):

// int value - значение, хранимое в узле,

// Node\*\* sons - массив длиной n, содержащий ссылки на сыновей.

// ================================================================

struct Node

{

int value;

Node\*\* sons;

Node(int val, int n) : value(val)

{

sons = new Node \*[n];

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

{

sons[i] = nullptr;

}

}

};

struct Tree

{

int n;

Node\* root;

Tree(int num) : n(num) {} // Этот конструктор будет вызываться с переданным числом и сохранять его в переменной n

};

// ===============================================================

// Функции для решения задачи

// ================================================================

// Рекурсивно обходит в глубину поддерево дерева tree с корнем поддерева node

// Проверяет, что все листья поддерева лежат в диапазоне глубины от корня [min, max]

// Возвращает строковое представление поддерева, если все листья находятся в диапазоне

// Возвращает пустую строку, если хотя бы один лист лежит за пределами диапазона

// level - текущий уровень глубины, на котором находится node (корень изначального дерева: level = 1)

string checkSubtree(Tree\* tree, Node\* node, int n, int min, int max, string path, int level) {

if ( node == nullptr )

return "";

bool childFree = true;

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

if (node -> sons[i] != nullptr)

childFree = false;

}

path += "\nNode " + to\_string(node -> value) + ": ";

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

path += ((node -> sons[i] == nullptr) ? "None" : to\_string(node -> sons[i] -> value)); //Добавляем к пути значение каждого сына узла, если он существует, или "None", если сын отсутствует.

if (i != (n - 1))

path += " and ";

}

if ( childFree ) {

if ( level < min || level > max )

return "";

return path;

}

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

if ( node -> sons[i] != nullptr )

{

path = checkSubtree(tree, node->sons[i], n, min, max, path, level + 1);

if ( path.empty() )

return "";

}

}

return path;

}

// Рекурсивно обходит в глубину (сверху вниз, слева направо) все узлы дерева tree с корнем node

// С помощью функции checkSubtree выполняет проверку, что все листья поддерева с корнем в текущем узле

// лежат в диапазоне [min, max]. Если условие выполняется, выводит структуру найденного поддерева в

// консоль.

// level - глубина, на которой находится node относительно корня изначального дерева tree

// (корень изначального дерева: level = 1)

bool findAllSubtreesInRange(Tree\* tree, Node\* node, int n, int min, int max, int level = 1) {

if ( node == nullptr ) {

return false;

}

bool found = false;

string path = checkSubtree(tree, node, n, min, max, "", level);

if ( !path.empty() ) {

cout << "////////////////////////////////////////////////////////////////////////////////////////////" << endl;

cout << "Subtree found:" << endl;

cout << path << endl << endl;

found = true;

}

if ( level == max )

return found;

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

if ( node->sons[i] != nullptr )

if (findAllSubtreesInRange(tree, node -> sons[i], n, min, max, level + 1))

found = true;

}

return found;

}

// Рекурсивная функция для ввода узлов дерева

// n - макс. количество потомков

// displayPrompts - показывать запросы ввода и подсказки в консоли

// (true - для ввода с клавиатуры, false - при чтении файла)

// Для каждого узла:

// Запрашивает ввод целочисленного значения (номера) узла node -> value

// n раз просит сообщить, имеется ли сын в соответствующей позиции в массиве: 1 - да, 0 - нет

// Если сын имеется, рекурсивно переходит к заполнению информации о нём

// Таким образом, выполняет построение дерева в глубину (сначала сверху вниз, затем слева направо, поднимаясь к корню)

// Возвращает Node\*

Node\* inputTree(istream& input, int n, bool displayPrompts = true) {

int rootValue;

if ( displayPrompts ) {

cout << "Введите значение корня дерева: ";

}

while ( !(input >> rootValue) ) {

cout << "Ошибка! Введите целое число: ";

input.clear();

input.ignore(32767, '\n');

}

Node\* root = new Node(rootValue, n);

int reply;

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

if ( displayPrompts ) {

cout << "Есть ли сын №" << (i + 1) << " у узла " << root->value << "? (1 - да, 0 - нет): ";

}

while ( !(input >> reply) || (reply != 0 && reply != 1) ) {

cout << "Ошибка ввода! Введите 0 или 1: ";

input.clear();

input.ignore(32767, '\n');

}

if(reply) {

root -> sons[i] = inputTree(input, n, displayPrompts);

}

}

return root;

}

// Запрашивает ввод максимального количества сыновей у каждого узла (арность) при создании дерева: tree -> n

// После чего запускает рекурсивную функцию inputTree для ввода узлов дерева, начяиная с корня tree -> root

// Возвращает Tree\* tree

Tree\* inputSize(istream& input, bool displayPrompts = true) {

int n;

if ( displayPrompts ) {

cout << "Введите значение n - арности дерева: ";

}

// Повторно заправшивает ввод, пока введено не целое число, которое больше нуля

while (!(input >> n) || n <= 0) {

cout << "Ошибка! Введите целое число > 0: ";

input.clear();

input.ignore(32767, '\n');

}

Tree\* tree = new Tree(n);

tree -> root = inputTree(input, n, displayPrompts);

return tree;

}

// Вывод дерева с корнем node и арностью n при обходе в глубину (сверху вниз, слева направо)

void printTreeDFS(Node \*node, int n) {

if ( node == nullptr )

return;

string sons;

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

sons += ((node -> sons[i] == nullptr) ? "None" : to\_string(node -> sons[i] -> value)); // Добавляет значение каждого сына узла node в строку sons,

// если сын узла существует.Если сын узла равен nullptr, то добавляется строка "None".

if (i != (n - 1))

sons += " and ";

}

cout << "Node "

<< node->value << ": " << sons << endl;

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

printTreeDFS(node -> sons[i], n);

}

}

// Вывод дерева при передачи структуры Tree\*, используя printTreeDFS(\*node, n)

void printTree(Tree \*tree) {

printTreeDFS(tree -> root, tree -> n);

}

// Чтобы окно командной строки не закрывалось при запуске .exe

void pause() {

string input;

cout << "\nВведите любой символ и нажмите Enter, чтобы выйти...";

cin >> input;

}

int main() {

setlocale(LC\_ALL, "ru\_RU.UTF-8");

system("chcp 65001");

string filename;

char choice;

bool inputValid = false;

do {

cout << "Выберите источник ввода (k - клавиатура, f - файл): ";

cin >> choice;

Tree\* tree;

if ( choice == 'k' ) {

tree = inputSize(cin);

inputValid = true;

}

else if ( choice == 'f' ) {

while ( true ) {

cout << "Введите имя файла: ";

cin >> filename;

ifstream inputFile(filename);

if (!inputFile.is\_open()) {

cerr << "Ошибка открытия файла! Повторите..." << endl;

continue;

}

tree = inputSize(inputFile, false);

inputFile.close();

inputValid = true;

break;

}

}

else

{

cout << "Некорректный выбор. Выберите k или f." << endl;

}

if ( inputValid ) {

cout << "Введённое дерево (DFS):" << endl;

printTree(tree);

int min, max;

cout << "Проограмма выполняет поиск всех поддеревьев, листья которых лежат" << endl

<< "в диапазоне глубины [min, max]. Глубина корня исходного дерева = 1." << endl;

cout << "Введите значение min: ";

// Повторно заправшивает ввод, пока введено не целое число, которое больше нуля

while ( !(cin >> min) ) {

cerr << "Ошибка! Введите целое число: ";

}

cout << "Введите значение max >= " << min << ": ";

// Повторно заправшивает ввод, пока введено не целое число, которое больше нуля

while ( !(cin >> max) || max < min ) {

cerr << "Ошибка! Введите целое число >= min (" << min << "): ";

}

if ( !findAllSubtreesInRange(tree, tree -> root, tree -> n, min, max) )

cout << "Для заданного дерева не было найдено поддеревьев, "

"все листья которых располагались бы в диапазоне [" << min << ", " <<

max << "]" << endl;

}

} while ( !inputValid );

pause();

}