**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Вычислительной техники**

отчет

**по лабораторной работе № 3**

**по дисциплине «Алгоритмы и структуры данных»**

Тема: **«Деревья»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3311 | Загуменнов И.М. Пасечный Л.В. |  |
| Преподаватель | Манирагена В. |  |

Санкт-Петербург

2024

**Цель работы**

Исследование алгоритмов для работы с двоичным деревом

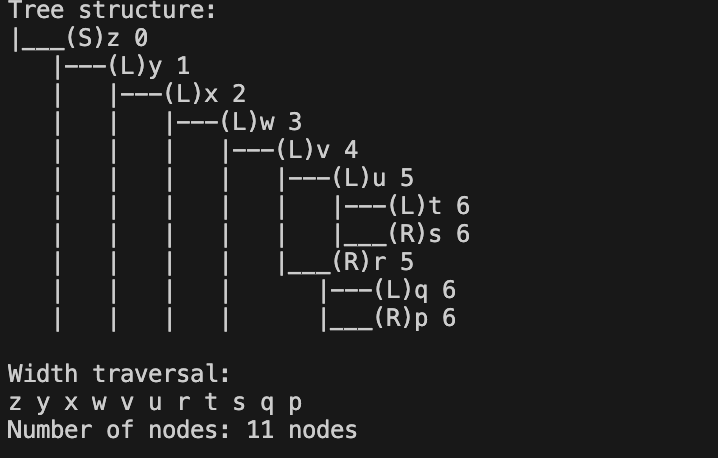
**Задание на работу с деревьями**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № варианта | Вид дерева | Разметка | Способ обхода | Что надо вычислить |
| 28 | Двоичное | Обратная | Внутренний | Количество предков у каждой из вершин |

**Обоснование выбора способа представления деревьев в памяти ЭВМ**

По лабораторным работам №1 и №2 было принято решение о реализации лабораторной работе без использования класса по причине экономии времени выполнения программы

**Тестовый пример**



Оценки временной сложности каждой функции обхода дерева, использованной в программе: создание дерева, обработка, вывод.

Создание дерева имеет сложность O(N)

Обработка дерева также имеет сложность O(N)

Вывод дерева аналогично имеет сложность O(N), т.к. при выводе мы проходим по каждому из узлов.

(где N — количество узлов в дереве)

**Ответы на вопросы**

Контрольные вопросы

1. Чем отличаются алгоритмы для разных способов обхода деревьев?

Алгоритмы обхода деревьев различаются порядком посещения узлов:

  - Прямой (pre-order): сначала обрабатывается корневой узел, затем левое поддерево и правое поддерево. Порядок: корень → левый ребенок → правый ребенок.

  - Внутренний (in-order): сначала обрабатывается левое поддерево, затем корневой узел, потом правое поддерево. Порядок: левый ребенок → корень → правый ребенок. Этот метод подходит для бинарных деревьев поиска, так как он выводит элементы в отсортированном порядке.

  - Обратный (post-order): сначала обрабатываются оба поддерева (левое и правое), а потом корневой узел. Порядок: левый ребенок → правый ребенок → корень.

  - Обход в ширину: посещаются все узлы на одном уровне дерева, а затем переходят к следующему уровню, начиная с корня.

Каждый из этих алгоритмов применяется в зависимости от задачи: поиск, сортировка или другие операции с деревьями.

2. Нужно ли сочетать ввод данных для построения дерева с клавиатуры с его обходом?

Не обязательно. Эти операции могут быть независимыми. Ввод данных можно реализовать на этапе построения дерева, а сам обход выполнять отдельно после того, как дерево построено. Однако в некоторых задачах может потребоваться ввод данных и немедленное выполнение обхода (например, если узлы добавляются динамически, и мы сразу хотим видеть изменения в дереве).

3. Можно ли считать применённые вами алгоритмы обхода дерева эффективными?

Да, алгоритмы, реализованные в коде (инфиксный обход и вычисление высоты дерева), считаются эффективными для большинства задач с бинарными деревьями. Время выполнения рекурсивных алгоритмов обхода — это \(O(n)\), где \(n\) — количество узлов в дереве. Каждый узел посещается один раз, что делает их достаточно производительными для стандартных бинарных деревьев.

4. Нужно ли создавать отдельные классы для узла и для дерева в целом, или можно ограничиться одним универсальным, рассматривая любой узел как корень некоторого поддерева?

Создание отдельных классов для узла и дерева — хороший объектно-ориентированный подход, так как он делает код более понятным и структурированным. Узел представляет базовый элемент дерева, а дерево управляет узлами и предоставляет методы для их обработки (например, добавление узлов, обходы и т.д.). Однако, если дерево имеет простую структуру, можно ограничиться одним классом, где каждый узел будет рассматриваться как корень поддерева. Это подходит для небольших программ, но в более сложных задачах отдельные классы улучшают читаемость и расширяемость кода.

**Выводы о результатах испытания алгоритмов обхода деревьев**

Программа работает корректно. Рекурсивные функции для обхода деревьев успешно применены.

**Приложение**

#include <iostream>

#include <cstring>

#include <cstdlib>

#include <ctime>

#include <limits>

#include <queue>

#include <stack>

using namespace std;

const int scale = 256;

class Node{

char d;

int pred;

Node \*lft;

Node \*rgt;

public:

Node() : lft(nullptr), rgt(nullptr) {}

~Node(){

if (lft) delete lft;

if (rgt) delete rgt;

}

friend class Tree;

};

class Tree{

Node \*root;

char num, maxnum; //счётчик тэков и максимальный тег

int maxrow; //максимальная глубина

Node\* MakeNode(int depth, const char\* position, int pred);

Node\* MakeRandomNode(int depth, int pred);

void printTree(Node\* node, string indent, bool isLast, char nodeType); //Красивый вывод

public:

Tree(char num, char maxnum, int maxrow);

~Tree();

void MakeTree();

void MakeRandomTree();

bool exist() { return root != nullptr; };

int BFS();

Node\* getRoot() {return root; }

void OutTree();

};

Tree::Tree(char nm, char mn, int mxr) :

num(nm), maxnum(mn), maxrow(mxr), root(nullptr) {}

Tree::~Tree(){

delete root;

}

/\*\*

\* @brief Function for check Users input.

\* @param depth глубина, на которой поставится символ.

\* @param position позиция символа(лево или право).

\*/

bool promtUser(int depth, const char\* position, char currentSymbol) {

char responce;

while (true){

cout << "Current symbol to add: " << currentSymbol << endl;

cout << "Do you want to fill this " << position << " node (level " << depth << ")? (1 - yes / 0 - no): ";

cin >> responce;

if (cin.fail() || (responce != '1' && responce != '0')){

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "Invalid input. Please enter 1 (yes) or 0 (no)." << endl;

}else{

break;

}

}

return responce=='1';

}

/\*\*

\* @brief Создание узла.

\* @param depth глубина дерева, на которой создаётся узел.

\* @param position Позиция (лево или право)

\* @param pred Кол-во предков у элемента.

\*/

Node\* Tree::MakeNode(int depth, const char\* position, int pred){

if (depth >= maxrow) return nullptr; // Проверка на максимальную глубину

Node \*v = nullptr;

if (num <= maxnum && promtUser(depth, position, num)){

v = new Node();

v->pred = pred;

v->d = maxnum--;

v->lft = MakeNode(depth+1, "left", v->pred + 1);

v->rgt = MakeNode(depth+1, "right", v->pred + 1);

}

return v;

}

/\*\*

\* @brief Создание рандомного узла

\* @param depth Глубина, на которой создаётся.

\* @param pred Кол-во предков у элемента.

\*/

Node\* Tree::MakeRandomNode(int depth, int pred){

if (num > maxnum || depth >= maxrow) return nullptr;

if (rand()%2 == 0) return nullptr;

Node \* v = new Node();

v->pred = pred;

v->d = maxnum--;

v->lft = MakeRandomNode(depth+1, v->pred+1);

v->rgt = MakeRandomNode(depth+1, v->pred+1);

return v;

}

/\*\*

\* @brief Создание корня и дерева.

\*/

void Tree::MakeTree(){

cout << "Generating tree manually..." << endl;

root = MakeNode(0, "root", 0);

}

/\*\*

\* @brief Создание рандомного корня и дерева.

\*/

void Tree::MakeRandomTree(){

cout << "Generating tree randomlly..." << endl;

root = MakeRandomNode(0, 0);

}

/\*\*

\* @brief Вывод дерева.

\*/

void Tree::OutTree(){

cout << "Tree structure: " << endl;

printTree(root, "", true, 'S'); //Начальный узел обозначен как корень 'S'

}

/\*\*

\* @brief Вывод дерева узел за узлом(рекурсивный метод)

\* @param node текущий узелж

\* @param indent отступ для того, чтобы не наезжали друг на друга

\* @param isLast является ли элемент последним.

\* @param nodeType тип узла(левый, правый, корень)

\*/

void Tree::printTree(Node\* node, string indent, bool isLast, char nodeType){

if (!node) return ;

cout << indent;

if (isLast){

cout << "|\_\_\_";

indent += " ";

}else{

cout << "|---";

indent += "| ";

}

//Печатаем узел с его типом

cout << "(" << nodeType << ")" << node->d << " " << node->pred << endl;

if (node->lft) printTree(node->lft, indent, false, 'L');

if (node->rgt) printTree(node->rgt, indent, true, 'R');

}

/\*\*

\* @brief Обход дерева в ширину.

\*/

int Tree::BFS(){

queue<Node\*> q;

int nodeCount = 0;

q.push(root);

while (!q.empty()){

Node\* n = q.front();

cout << n->d << " ";

nodeCount++;

q.pop();

if(n->lft) q.push(n->lft);

if(n->rgt) q.push(n->rgt);

}

return nodeCount;

}

int main(){

int choice = -1;

while (true){

cout << "How would you like to fill the tree& (1 - manually, 0 - randomlly, 2 - exit): ";

cin >> choice;

if (cin.fail() || (choice != 1 && choice != 0 && choice != 2)){

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "Invalid input. Please input 1, 0 or 2. " << endl;

continue;

}

if (choice == 2) break;

Tree Tr('a', 'z', 7);

srand(time(nullptr));

if (choice == 1){

Tr.MakeTree();

}else if (choice == 0){

Tr.MakeRandomTree();

}

if (Tr.exist()){

Tr.OutTree();

cout << '\n' << "Width traversal: "<< endl;

int n = Tr.BFS();

cout << "\nNumber of nodes: " << n << " nodes" << endl;

} else{

cout << "The tree is empty!";

}

cout << '\n' << "=========== END ==============\n";

}

return 0;

}