

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. 1306

Пестерев В.А.

Преподаватель

Колинько П.Г.

Санкт-Петербург

2022

Введение

Цель работы: исследование алгоритмов для работы с троичным деревом.





Задание (вариант №27)

Написать и отладить программу для работы с троичными деревьями. Найти высоту среднего поддерева для корня с применением обхода в ширину.

Способ представления деревьев в памяти ЭВМ

Дерево в памяти будет представлено в виде списка, элементами которого будут являться узлы данного дерева. Так же дерево и его узлы являются объектами соответствующих классов, а обходы — функциями-членами. Согласно результатам экспериментов, проведенных в ходе выполнения Лабораторных работ №1 и №2, данный способ представления будет являться оптимальным, так как предоставляет возможность обрабатывать динамические деревья с заранее неизвестным количеством вершин.

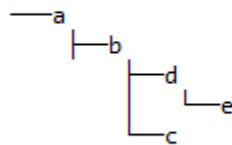
Вывод дерева происходит горизонтально, в порядке от отца к сыну (сыновья находятся ниже и правее отца), в соответствии со следующими обозначениями:

- «» - корень;
- «» - левый сын;
- «» - средний сын;
- «» - правый сын.

Результаты эксперимента

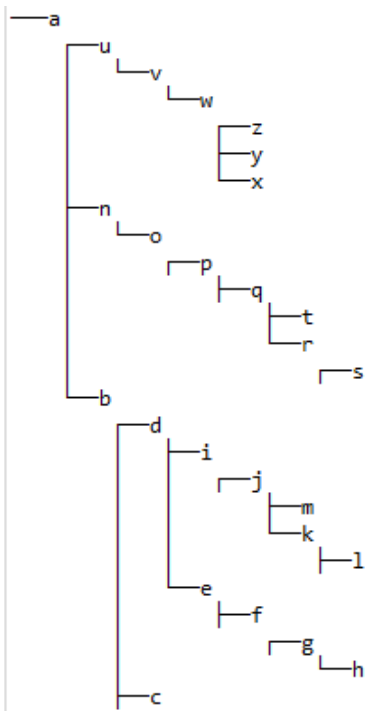
В качестве результатов эксперимента на Рисунках 1-3 представлены примеры прогона программы с ручным вводом и случайной генерацией деревьев.

```
Node (a,0)1/0: 1
Node (b,1)1/0: 0
Node (b,1)1/0: 1
Node (c,2)1/0: 1
Node (d,3)1/0: 0
Node (d,3)1/0: 0
Node (d,3)1/0: 0
Node (d,2)1/0: 1
Node (e,3)1/0: 1
Node (f,4)1/0: 0
Node (f,4)1/0: 0
Node (f,4)1/0: 0
Node (f,3)1/0: 0
Node (f,3)1/0: 0
Node (f,2)1/0: 0
Node (f,1)1/0: 0
```



Обход графа в ширину: a_b_c_d_e_ Количество узлов: 5
Длина среднего поддеревя: 3

Рисунок 1: Тестовый пример: ввод дерева с клавиатуры



Обход графа в ширину: a_b_n_u_c_d_o_v_e_i_p_w_f_j_q_x_y_z_g_k_m_r_t_h_l_s_ Количество узлов: 26
 Длина среднего поддеревя: 6

Рисунок 2: Результат прогона программы с генерацией случайного дерева



Обход графа в ширину: a_b_c_h_d_f_i_l_w_e_g_j_m_r_t_x_k_n_o_s_u_y_p_v_z_q_ Количество узлов: 26
Длина среднего поддереза: 0

Рисунок 3: Дерево с отсутствующим средним поддерезом

Оценка временной сложности

Так как алгоритмы создания, обработки и вывода дерева сводятся к тому, чтобы перебрать все вершины дерева, то для дерева на n вершин сложность составит $O(n)$ и будет линейной.

Вывод

В результате выполнения работы были изучены способы использования алгоритмов обхода деревьев на языке C++ и получены практические навыки в работе с ними, а также была написана программа, способная вычислить длину среднего поддереза троичного дерева с применением обхода графа в ширину.

Список литературы

Колинько П. Г. Алгоритмы и структуры данных. Часть 1. Пользовательские структуры данных: Конспект лекций. Вып. 2201. — СПб.: СПбГЭТУ «ЛЭТИ», 2021. — 565с.

Колинько П. Г. Пользовательские структуры данных: Методические указания по дисциплине «Алгоритмы и структуры данных, часть 1». — СПб.: СПбГЭТУ «ЛЭТИ», 2022 — 64 с. (вып.2209)

Шилдт Г. С++ шаг за шагом. Пер. С англ. М.: — Экон Паблишерс, 2009. — 640с.: ил.

Страуструп Б. Язык программирования С++. Специальное издание. Пер. с англ. — М.: Изд-во Бином, 2015 — 1136 с.: ил.

Приложение

Исходные текст программы main.cpp:

```
//=====
// Name      : lab3.cpp
// Author    : Pesterev
// Version   :
// Copyright  :
// Description: Алгоритмы для работы с троичным деревом
//=====

#include <iostream>
#include <queue>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#include <locale>
using namespace std;

class Node {
    char d;
    Node *lft, *mdl, *rgt;
    int deep = 0;
public:
    Node() :
        lft(nullptr), mdl(nullptr), rgt(nullptr) {
    }
    ;
    ~Node() {
        if (lft)
            delete lft;
        if (mdl)
            delete mdl;
        if (rgt)
            delete rgt;
    }

    friend class Tree;
};

class Tree {
    Node *root;
    char num, maxnum;
    int maxrow;

    Node* makeNode(int depth);
    Tree(const Tree&);
    Tree(Tree&&);

    Tree operator =(Tree&&) const = delete;

public:
    Tree(char num, char maxnum, int maxrow);
    ~Tree();
    void makeTree() {
        root = makeNode(0);
    }
    bool exist() {
        return root != nullptr;
    }
    int BFS();
};
```

```

        void printBT();
        void printBT(const string &prefix, const Node *node, int dim, bool
hasSib);
        int midLen();
        int heigh(Node *nd);
};

Tree::Tree(char nm, char mnm, int mxr) :
        num(nm), maxnum(mnm), root(nullptr), maxrow(mxr) {}

Tree::~Tree() {
        delete root;
}

Node* Tree::makeNode(int depth) {
        Node *v = nullptr;

        if (num <= maxnum) {
                int Y = (depth < rand() % maxrow + 1);
                /*
                cout << "Node (" << num << ', ' << depth << ")1/0: ";
                cin >> Y;
                */

                if (Y) { // создание узла, если Y = 1
                        v = new Node;
                        v->d = num++;
                        if (depth < maxrow) {
                                v->lft = makeNode(depth + 1);
                                v->mdl = makeNode(depth + 1);
                                v->rgt = makeNode(depth + 1);
                        }
                }
        }
        return v;
}

int Tree::BFS() {
        int count = 0;
        queue<Node*> Q; //создание очереди указателей на узлы
        Q.push(root); // поместить в очередь корень дерева
        while (!Q.empty()) //пока очередь не пуста
        {
                Node *v = Q.front();
                Q.pop(); // взять из очереди,
                cout << v->d << ' ';
                ++count; // выдать тег, счёт узлов
                if (v->lft)
                        Q.push(v->lft); // Q <- (левый сын)
                if (v->mdl)
                        Q.push(v->mdl);
                if (v->rgt)
                        Q.push(v->rgt); // Q <- (правый сын)
        }
        return count;
}

void Tree::printBT(const string &prefix, const Node *node, int dim,
        bool hasSib) {
        if (node != nullptr) {
                cout << prefix;
                switch (dim) {

```



```

        case 0: {
            cout << ("└─");
            cout << node->d << endl;
            break;
        }
        case 1: {
            cout << ("├─");
            cout << node->d << endl;
            break;
        }
        case 2: {
            cout << ("└─");
            cout << node->d << endl;
            break;
        }
        case 3: {
            cout << ("──");
            cout << node->d << endl;
            break;
        }
    }
    printBT(prefix + (hasSib ? "│" : " "), node->rht, 0,
            (node->mdl != nullptr) || (node->lft != nullptr));
    printBT(prefix + (hasSib ? "│" : " "), node->mdl, 1,
            node->lft != nullptr);
    printBT(prefix + (hasSib ? "│" : " "), node->lft, 2, false);
}

}

void Tree::printBT() {
    printBT("", root, 3, false);
}

/*int Tree::heigh(Node *nd) {
    int len = -1;
    if (nd) {
        if (nd->lft != nullptr || nd->mdl != nullptr || nd->rht !=
            nullptr) {
                len = max(max(heigh(nd->lft), heigh(nd->mdl)), heigh(nd-
                >rht));
            } else
                len = 0;
        }
        return len + 1;
    }
}*/

int Tree::heigh(Node *nd) {
    int len = 0;
    queue<Node*> Q; //создание очереди указателей на узлы
    Q.push(nd); // поместить в очередь корень дерева
    if (nd)
        nd->deep = 1;
    while (!Q.empty()) //пока очередь не пуста
    {
        Node *v = Q.front();
        Q.pop(); // взять из очереди,
        if (v->lft) {
            Q.push(v->lft); // Q <- (ЛЕВЫЙ СЫН)
            v->lft->deep = v->deep + 1;
        }
        if (v->mdl) {
            Q.push(v->mdl);
            v->mdl->deep = v->deep + 1;
        }
    }
}

```

```

    }
    if (v->rgt) {
        Q.push(v->rgt); // Q <- (правый сын)
        v->rgt->deep = v->deep + 1;
    }

    len = max(len, v->deep);
}
return len;
}

int Tree::midLen() {
    int len = 0;
    if (root && root->mdl) {
        len = heigh(root->mdl);
    }
    return len;
}

int main() {
    int n = 0;
    Tree Tr('a', 'z', 8);
    srand(time(nullptr));
    setlocale(LC_ALL, "Russian");

    Tr.makeTree();
    if (Tr.exist()) {
        Tr.printBT();
        cout << '\n' << "Обход графа в ширину: ";
        n = Tr.BFS();
        cout << " Количество узлов: " << n << endl;
        cout << "Длина среднего поддерева: " << Tr.midLen();
    } else
        cout << "Пусто!";

    return 0;
}

```