**МИНОБРНАУКИ РОССИИ**

**Санкт–Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

отчет

**по лабораторной работе №4**

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

Тема: Деревья

|  |  |  |
| --- | --- | --- |
| Студент гр. 7381 |  | Ильясов А.В. |
| Преподаватель |  | Фирсов М.А. |

Санкт–Петербург

2018

**Задание**

Вариант 2(а–г):

Для заданного бинарного дерева b типа BT с произвольным типом элементов:

а) определить максимальную глубину дерева b, т. е. число ветвей в самом длинном из путей от корня дерева до листьев;

б) вычислить длину внутреннего пути дерева b, т. е. сумму по всем узлам длин путей от корня до узла;

в) напечатать элементы из всех листьев дерева b;

г) подсчитать число узлов на заданном уровне n дерева b (корень считать узлом 1–го уровня);

Реализация дерева должна быть основана на динамической памяти.

**Пояснение задания**

Наиболее важным типом деревьев являются бинарные деревья. Удобно дать следующее формальное определение. Бинарное дерево – конечное множество узлов, которое либо пусто, либо состоит из корня и двух непересекающихся бинарных деревьев, называемых правым поддеревом и левым поддеревом.

Например, бинарное дерево, изображенное на рис. 1, имеет скобочное представление:

(*a* (*b* (*d* Λ (*h* Λ Λ)) (*e* Λ Λ)) (*c* (*f* (*i* Λ Λ) (*j* Λ Λ)) (*g* Λ (*k* (*l* Λ Λ) Λ)))).

Рис. 1 – Бинарное дерево

*a*

*c*

*b*

*f*

*d*

*h*

*e*

*j*

*g*

*i*

*k*

*l*

Определим скобочное представление бинарного дерева (БД):

< БД > ::= < пусто > | < непустое БД >,

< пусто > ::= ᴧ,

< непустое БД > ::= ( < корень > < БД > < БД > ).

Задача каждого из пунктов сводится к рекурсивному обходу дерева, заданного скобочным выражением, и выполнение определенных действий.

**Описание алгоритмов**

а) По определению, корень дерева имеет глубину 1, значит глубина непустого дерева не меньше 1. При переходе в левое или правое поддерево, глубина дерева увеличивается на 1, то есть равна 2. При дальнейшем переходе к поддеревьям текущих корней, глубина увеличивается на 1. И это происходит до тех пор, пока мы не натыкаемся на лист – узел дерева, не имеющий детей.

При каждом вызове метода подсчета глубины производится проверка на наличие поддеревьев, и если поддеревья есть, то рекурсивно вызывается этот же метод для поддеревьев с прибавлением единицы к промежуточному значению глубины деревьев, являющихся поддеревьями заданного. После этого выбирается наибольшее значение глубины левого или правого поддерева, которое возвращается, как результат выполнения метода, и также будет сравнен со значением, полученным в другом поддереве, и также будет сравнен, только на предыдущем шаге рекурсии.

б) Чтобы посчитать длину пути в дереве, необходимо пройти по всем поддеревьям и с каждым шагом прибавлять единицу к длине.

При каждом рекурсивном дереве создается 2 переменные, отвечающие за длину пути в левом и правом поддереве, инициализирующиеся 0. Это сделано исключительно для подробного вывода работы программы. Далее вызывается этот же метод для каждого поддерева, если оно есть. И так продолжается до тех пор, пока не встречается лист, и тогда длины путей в поддеревьях складываются и возвращаются с последующим складыванием с полученным значением в другом поддереве на предыдущем шаге рекурсии.

в) Пока текущий просматриваемый узел не лист, происходит переход в его левое или правое поддерево. Когда встречается лист, значение этого узла записывается в выходную строку, также исключительно для подробного вывода работы программы, которая потом выводится на экран.

г) Первоначально производится проверка на введенное значение, которое должно быть строго положительным и не превышать максимальную глубину дерева, посчитанную в пункте а). Если значение удовлетворяет этим условиям, то это значение передается, как аргумент в метод, в котором происходит такой же обход дерева, который описывался в предыдущих пунктах, с тем отличием, что в этом методе при каждом погружении на следующий уровень дерева уменьшается значение получаемого аргумента на единицу. При достижении нужного уровня глубины, значение аргумента равно 1, так как корень дерева лежит на первом уровне. Если на этом уровне в поддереве есть узел, то счетчик узлов увеличивается на единицу. Таким образом при обходе всего дерева, будут обнаружены все узлы на заданном уровне.

**Описание функций**

void print\_tabs(size\_t tabs);–функция, печатающая необходимое количество символов табуляции.

size\_t tabs – количество выводимых символов табуляции.

*Возвращаемое значение*: функция ничего не возвращает.

enum side\_t {LEFT, RIGHT}; - перечисление, используемое для выбора стороны поддерева при создании бинарного дерева.

**Описание класса** BinaryTree**:**

**поля:**

char value; – значение корня.

size\_t depth; – глубина дерева

BinaryTree \*left; – указатель на левое поддерево.

BinaryTree \*root; – указатель на родителя.

BinaryTree \*right; – указатель на правое поддерево.

конструкторы:

BinaryTree(); – стандартный конструктор. Инициализирует указатели left, right, root нулевыми указателями. Используется при создании поддеревьев.

BinaryTree(std::string &string); – конструктор, передающий выходную строку в метод чтения и создания дерева.

~BinaryTree(); – деструктор, удаляющий узлы дерева с конца.

**приватные методы:**

void read\_\_binary\_tree(std::string &string); – основной метод чтения входной строки и создания дерева по его скобочной записи. В этом методе происходит анализ корректности строки, а также ее разделение для последующего перенаправления в методы создания дерева.

std::string &string – строка, содержащая скобочную запись дерева.

*Возвращаемое значение*: метод ничего не возвращает.

void record\_to\_root(char value); – метод, записывающий значение value в узел.

char value – символ, который будет записан в узел, как его значение. Должен быть либо цифрой, либо буквой.

*Возвращаемое значение*: метод ничего не возвращает.

void create\_tree\_branches(std::string &string, side\_t side); – метод, создающий поддеревья в дереве. В нем выделяется память под дерево и вызывается read\_\_binary\_tree для дальнейшего создания дерева.

std::string &string – строка, содержащая скобочную запись дерева.

side\_t side – переменная, определяющая сторону создаваемого поддерева.

*Возвращаемое значение*: метод ничего не возвращает.

size\_t get\_depth(size\_t lvl); – метод, вычисляющий глубину заданного дерева с одновременным выводом процесса работы.

size\_t lvl – количество отступов, необходимое для читаемого вывода процесса работы метода.

*Возвращаемое значение*: глубина дерева в формате целого беззнакового.

size\_t get\_path\_length(size\_t lvl); – метод, вычисляющий длину пути в заданном дереве с одновременным выводом процесса работы.

size\_t lvl – количество отступов, необходимое для читаемого вывода процесса работы метода.

*Возвращаемое значение*: длина пути в дереве в формате целого беззнакового.

void print\_leaves(size\_t lvl, std::string &leaves); – метод, печатающий все листья в заданном дереве с одновременным выводом процесса работы.

size\_t lvl – количество отступов, необходимое для читаемого вывода процесса работы метода.

std::string &leaves – строка, в которую помещаются все найденные листья.

*Возвращаемое значение*: метод ничего не возвращает.

size\_t get\_nodes\_number(int depth\_lvl, size\_t lvl); – метод, вычисляющий количество узлов на заданном уровне в дереве с одновременным выводом процесса работы.

int depth\_lvl – глубина, на которой нужно посчитать количество узлов. Вводится пользователем.

size\_t lvl – количество отступов, необходимое для читаемого вывода процесса работы метода.

*Возвращаемое значение*: количество узлов на заданном уровне в формате целого беззнакового.

**публичные метода:**

size\_t get\_depth(); – метод, вызывающий перегруженный метод get\_depth(size\_t lvl) с начальным значением lvl равным 1. Именно этот метод вызывает пользователь.

*Возвращаемое значение*: то же самое значение, что и перегруженный метод.

size\_t get\_path\_length(); – метод, вызывающий перегруженный метод get\_path\_length(size\_t lvl) с начальным значением lvl равным 1. Именно этот метод вызывает пользователь.

*Возвращаемое значение*: то же самое значение, что и перегруженный метод.

void print\_leaves(); – метод, вызывающий перегруженный метод print\_leaves(size\_t lvl) с начальным значением lvl равным 1. Именно этот метод вызывает пользователь.

*Возвращаемое значение*: метод ничего не возвращает.

size\_t get\_nodes\_number(int depth\_lvl); – метод, вызывающий перегруженный метод get\_nodes\_number(int depth\_lvl, size\_t lvl) с начальным значением lvl равным 1. Именно этот метод вызывает пользователь.

int depth\_lvl - глубина, на которой нужно посчитать количество узлов. Вводится пользователем.

*Возвращаемое значение*: то же самое значение, что и перегруженный метод.

**Тестирование**

Для проверки работоспособности программы был создан скрипт(см. ПРИЛОЖЕНИЕ ) для автоматического ввода и вывода тестовых данных:

correct test1.txt: (a(b)(c));

correct test2.txt: (a(b#(c))(d(e)(f(g))));

correct test3.txt: (a(b(c(d)(e))(f(g)(h)))(i(j(k)(l))(m(n)(o))));

correct test4.txt: (a(b(c(d)(e)))(i#(m(n)(o))));

correct test5.txt: (ab(cd)ef(gh(ij))kl(mn(op(qr))));

correct test6.txt: (a(b(c(d(e(f(g))))))(h#(i#(j#(k#(l#(m)))))));

incorrect test1.txt: q(d)(e);

incorrect test2.txt: (a(b#(c))(d(e)(f())));

incorrect test3.txt: (a(b#(c))(d(e)(f(g))))).

Результаты тестирования сохраняются в файл testsresult.txt.

Ниже представлены результаты тестирования.

|  |  |
| --- | --- |
| Скобочная запись дерева | Результат тестирования |
| (a(b)(c)) | The depth of this tree is equal to 2  the length of the path in this tree is equal to 2  All the leaves of this tree: b c  At level 2 is 2 nodes |
| (a(b#(c))(d(e)(f(g)))) | The depth of this tree is equal to 4  the length of the path in this tree is equal to 6  All the leaves of this tree: c e g  At level 3 is 3 nodes |
| (a(b(c(d)(e))(f(g)(h)))(i(j(k)(l))(m(n)(o)))) | The depth of this tree is equal to 4  the length of the path in this tree is equal to 14  All the leaves of this tree: d e g h k l n o  At level 3 is 4 nodes |
| (a(b(c(d)(e)))(i#(m(n)(o)))) | The depth of this tree is equal to 4  the length of the path in this tree is equal to 8  All the leaves of this tree: d e n o  At level 4 is 4 nodes |
| (a(b(c(d(e(f(g))))))(h#(i#(j#(k#(l#(m))))))) | The depth of this tree is equal to 7  the length of the path in this tree is equal to 12  All the leaves of this tree: g m  At level 3 is 2 nodes |
| q(d)(e) | --Error! Tree elements must be enclosed in parentheses-- |
| (a(b#(c))(d(e)(f()))) | --Error! Invalid character ' '-- |
| (a(b#(c))(d(e)(f(g))))) | --Error! Expected brackets or '#', but was ')'-- |

Разберем более подробно работу программы на примере первого теста:

Было введено (a(b)(c)).

а) подсчет глубины

Первым делом инициализируются единицей переменные для глубины левого и правого поддеревьев. У корня есть и левое, и правое поддерево, поэтому переменные увеличиваются на 1. Но корни поддеревьев являются листьями, поэтому вызовы методов завершаются и происходит сравнение полученных значений. Но 2 = 2, значит глубина заданного дерева = 2.

б) длина пути

Первым делом инициализируются нулем переменные для длины пути в левом и правом поддеревьях. У корня есть и левое, и правое поддерево, поэтому переменные увеличиваются на 1 и на значение, возвращаемое рекурсивным вызовом метода для каждого из поддеревьев. Но корни поддеревьев являются листьями, поэтому вызовы методов возвращают 0, что дает в итоге длину пути в левом и правом поддереве по 1. Значит суммарный путь имеет длину 2.

в) вывод листьев

У корня есть и левое, и правое поддерево, поэтому вызывается метод для поддеревьев, в которых корни уже являются листьями со значениями b и с. И так как для левого поддерева метод вызывается раньше, то сначала в выходную строку записывается левый лист - b, и потом уже правый - с. В итоге получается строка "b c".

г) подсчет узлов на заданном уровне

При тестировании выбор уровня производится рандомно в пределах от 1 до глубины дерева включительно. В данном случае это 2. При первом вызове метода с аргументом depth\_lvl = 2 инициализируется нулем переменная, хранящая количество узлов на уровне 2. У корня есть и левое, и правое поддерево, поэтому переменная увеличивается на значение возвращаемое рекурсивным вызовом метода для поддеревьев, но уже с аргументом depth\_lvl = 1. Условие равенства аргумента единице является условием выхода из рекурсии, и завершается он возвращением 1. Таким образом сначала переменная увеличивается на 1, за счет левого поддерева, а потом еще на 1, за счет правого. В итоге получаем количество узлов на 2 уровне равное 2.

**Выводы**

В процессе выполнения лабораторной работы были поли получены знания и навыки по ООП в С++. Были изучены бинарные деревья, повторена работа с рекурсивным функциям, bash–скриптам и автоматизации тестирования. Работа была написана на C/C++.

Приложение А

Код main.cpp

#include <iostream>

#include <string>

#include <algorithm>

#include <ctime>

#include "BTree.hpp"

#define TEST

int main() {

    std::string input\_data;

    std::cout << "Enter the bracket tree view: ";

    std::getline(std::cin, input\_data);

    input\_data.erase(remove\_if(input\_data.begin(), input\_data.end(), isspace), input\_data.end());

    BinaryTree binary\_tree(input\_data);

    size\_t depth = binary\_tree.get\_depth();

    std::cout << "The depth of this tree is equal to " << depth << std::endl;

    std::cout << std::endl;

    size\_t path\_length = binary\_tree.get\_path\_length();

    std::cout << "the length of the path in this tree is equal to " << path\_length << std::endl;

    std::cout << std::endl;

    binary\_tree.print\_leaves();

    std::cout << std::endl;

    int lvl;

    #ifndef TEST

        std::cout << "Enter a number between 1 and " << depth << ": ";

        std::cin >> lvl;

    #else

        srand(time(0));

        lvl = depth - rand() % depth;

    #endif

    size\_t nodes\_on\_lvl = binary\_tree.get\_nodes\_number(lvl);

    std::cout << "At level " << lvl << " is " << nodes\_on\_lvl << " nodes" << std::endl;

return 0;

}

Приложение Б

ФАЙЛ Btree.hpp

#ifndef \_\_BTREE\_HPP\_\_

#define \_\_BTREE\_HPP\_\_

enum side\_t {LEFT, RIGHT};

class BinaryTree {

private:

    char value;                             // Переменная для хранения значения в корне

    size\_t depth;                           // Переменная для хранения глубины дерева

    BinaryTree \*left;

    BinaryTree \*root;

    BinaryTree \*right;

    void read\_\_binary\_tree(std::string &string); // Методы четния

    void record\_to\_root(char value);             // входной строки и

    void create\_tree\_branches(std::string &string, side\_t side);

// создания дерева

    size\_t get\_depth(size\_t lvl);

    size\_t get\_path\_length(size\_t lvl);

    void print\_leaves(size\_t lvl, std::string &leaves);

    size\_t get\_nodes\_number(int depth\_lvl, size\_t lvl);

public:

    BinaryTree();

    BinaryTree(std::string &string);

    size\_t get\_depth();                          // Метод для пункта а)

    size\_t get\_path\_length();                    // Метод для пункта б)

    void print\_leaves();                         // Метод для пункта в)

    size\_t get\_nodes\_number(int depth\_lvl);      // Метод для пункта г)

    ~BinaryTree();

};

#endif

Приложение В

ФАЙЛ btree.cpp

#include <iostream>

#include <cstdlib>

#include <string>

#include <algorithm>

#include <cmath>

#include "BTree.hpp"

#define TEST

void print\_tabs(size\_t tabs) {

for (size\_t count = 0; count < tabs; count++)

std::cout << ".\t";

}

BinaryTree::BinaryTree() {

left = NULL;

root = NULL;

right = NULL;

}

BinaryTree::BinaryTree(std::string &string) {

read\_\_binary\_tree(string);

}

void BinaryTree::read\_\_binary\_tree(std::string &string) {

if (string[0] == '(' && string[string.length()-1] == ')') {

string = string.substr(1, string.length()-2);

record\_to\_root(string[0]);

string = string.substr(1, string.length()-1);

if (!string.length())

return;

if (string[0] == '(') {

size\_t index = 0;

size\_t brackets\_count = 0;

do {

if (string[index] == '(')

brackets\_count++;

if (string[index] == ')')

brackets\_count--;

index++;

} while (brackets\_count > 0);

std::string substring\_left = string.substr(0, index);

create\_tree\_branches(substring\_left, LEFT);

string = string.substr(index);

}

else if (string[0] == '#') {

string = string.substr(1, string.length()-1);

}

else {

std::cout << "--Error! Expected brackets or '#', but was '" << string[0] << "'--" << std::endl;

exit(0);

}

if (string.length() == 0)

return;

if (string[0] == '(') {

create\_tree\_branches(string, RIGHT);

}

else if (string[0] == '#') {

string = string.substr(1, string.length()-1);

}

else {

std::cout << "--Error! Expected brackets or '#', but was '" << string[0] << "'--" << std::endl;

exit(0);

}

}

else {

std::cout << "--Error! Tree elements must be enclosed in parentheses--" << std::endl;

exit(0);

}

}

void BinaryTree::record\_to\_root(char value) {

if (isdigit(value) || isalpha(value)) {

this->value = value;

left = NULL;

right = NULL;

}

else {

std::cout << "--Error! Invalid character '" << value << "'--" << std::endl;

exit(0);

}

}

void BinaryTree::create\_tree\_branches(std::string &string, side\_t side) {

if (side == LEFT) {

left = new BinaryTree();

left->root = this;

left->read\_\_binary\_tree(string);

}

else {

right = new BinaryTree();

right->root = this;

right->read\_\_binary\_tree(string);

}

}

size\_t BinaryTree::get\_depth() {

#ifndef TEST

std::cout << "Call depth calculation method" << std::endl;

#endif

size\_t depth = get\_depth(1);

this->depth = depth;

return depth;

}

size\_t BinaryTree::get\_depth(size\_t lvl) {

#ifndef TEST

print\_tabs(lvl);

std::cout << "method call. Current root - " << value << ". Current depth is " << lvl << std::endl;

#endif

size\_t left\_depth = 1;

size\_t right\_depth = 1;

if (left) left\_depth += left->get\_depth(lvl+1);

if (right) right\_depth += right->get\_depth(lvl+1);

// каждый раз выбирается максимальная глубина поддеревьев

size\_t max\_depth = std::max(left\_depth, right\_depth);

#ifndef TEST

print\_tabs(lvl);

std::cout << "exit method. compare the depth of the subtrees: left = " << left\_depth-1 << ", right = " << right\_depth-1

<< ". Choose more, that is " << max\_depth-1 << std::endl;

#endif

return max\_depth;

}

size\_t BinaryTree::get\_path\_length() {

#ifndef TEST

std::cout << "Call the method of calculating the length of the path in this tree" << std::endl;

#endif

return get\_path\_length(1);

}

size\_t BinaryTree::get\_path\_length(size\_t lvl) {

#ifndef TEST

print\_tabs(lvl);

std::cout << "method call. Current root - " << value << std::endl;

#endif

if (!left && !right) {

#ifndef TEST

print\_tabs(lvl);

std::cout << "exit method. It's a leave" << std::endl;

#endif

return 0;

}

else {

size\_t left\_length = 0;

size\_t right\_length = 0;

if (left) left\_length += 1 + left->get\_path\_length(lvl+1);

if (right) right\_length += 1 + right->get\_path\_length(lvl+1);

size\_t length = left\_length + right\_length;

#ifndef TEST

print\_tabs(lvl);

std::cout << "exit method. look at the length of the subtree paths: left = " << left\_length << ", right = " << right\_length

<< ". total path is " << left\_length << " + " << right\_length << " = " << length << std::endl;

#endif

return length;

}

}

void BinaryTree::print\_leaves() {

std::string leaves = "";

#ifndef TEST

std::cout << "Call the method of printing the leaves of this tree" << std::endl;

#endif

print\_leaves(1, leaves);

std::cout << "All the leaves of this tree: " << leaves << std::endl;

}

void BinaryTree::print\_leaves(size\_t lvl, std::string &leaves) {

#ifndef TEST

print\_tabs(lvl);

std::cout << "method call. Current root - " << value << std::endl;

#endif

if (!left && !right) {

#ifndef TEST

print\_tabs(lvl);

std::cout << "exit method. It's a leave" << std::endl;

#endif

leaves += value;

leaves += " ";

}

else {

if (left) left->print\_leaves(lvl+1, leaves);

if (right) right->print\_leaves(lvl+1, leaves);

#ifndef TEST

print\_tabs(lvl);

std::cout << "exit method. look at the leaves found: " << leaves << std::endl;

#endif

}

}

size\_t BinaryTree::get\_nodes\_number(int depth\_lvl) {

if (depth\_lvl < 1) {

std::cout << "--Error! Expected number not less than 1--" << std::endl;

exit(0);

}

if (depth\_lvl > depth) {

// аргумент не должен превышать максимальную глубину дерева

std::cout << "--Error! The argument entered is greater than the tree depth--" << std::endl;

exit(0);

}

#ifndef TEST

std::cout << "Call the method that counts the number of nodes at a given level" << std::endl;

#endif

return get\_nodes\_number(depth\_lvl, 1);

}

size\_t BinaryTree::get\_nodes\_number(int depth\_lvl, size\_t lvl) {

#ifndef TEST

print\_tabs(lvl);

std::cout << "method call. Current lvl is " << lvl << std::endl;

#endif

if (depth\_lvl == 1) {

// необходимый уровень достигается, когда lvl уменьшается до 1

#ifndef TEST

print\_tabs(lvl);

std::cout << "exit method. reached the desired level" << std::endl;

#endif

return 1;

}

else {

size\_t nodes\_number = 0;

if (left) nodes\_number += left->get\_nodes\_number(depth\_lvl-1, lvl+1);

if (right) nodes\_number += right->get\_nodes\_number(depth\_lvl-1, lvl+1);

#ifndef TEST

print\_tabs(lvl);

std::cout << "exit method. found " << nodes\_number << " nodes" << std::endl;

#endif

return nodes\_number;

}

}

BinaryTree::~BinaryTree() {

if (left)

delete left;

if (right)

delete right;

}

Приложение Г

makefile

all: main.o BTree.o

    g++ main.o BTree.o -o lab4

main.o: ./Source/main.cpp

    g++ -c ./Source/main.cpp

BTree.o: ./Source/BTree.hpp

    g++ -c ./Source/BTree.cpp

clean:

    rm main.o BTree.o

Приложение д

файл runtests.sh

#!/bin/bash

if test ! -f "lab4" ; then

    make make clean

fi

if test -f "testsresult.txt"; then

    rm testsresult.txt

fi

touch testsresult.txt

for i in $(ls ./Tests/correct); do

    echo "running correct $i: $(cat Tests/correct/$i | more)"

        sleep 0.1s

    echo "correct "$i"" >> testsresult.txt

    echo "test data: $(cat Tests/correct/$i | more)" >> testsresult.txt

    echo "result: " >> testsresult.txt

    ./lab4 < ./Tests/correct/$i >> testsresult.txt

    echo "###################################################" >> testsresult.txt

done

for i in $(ls ./Tests/incorrect); do

    echo "running incorrect $i: $(cat Tests/incorrect/$i | more)"

        sleep 0.1s

    echo "incorrect "$i"" >> testsresult.txt

    echo "test data: $(cat Tests/incorrect/$i | more)" >> testsresult.txt

    echo "result: " >> testsresult.txt

    ./lab4 < ./Tests/incorrect/$i >> testsresult.txt

    echo "###################################################" >> testsresult.txt

done

sleep 0.2s

echo "test results are saved in testsresult.txt"

rm lab4

Приложение е

файл run.sh

#!/bin/bash

if test ! -f "lab4"; then

    make

    make clean

fi

clear

./lab4

make clean