

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ
по лабораторной работе №3
по дисциплине «Алгоритмы и структуры данных»
Тема: Деревья

Студент гр. 9304

Ковалёв П. Д.

Преподаватель

Филатов А. Ю.

Санкт-Петербург

2020

Цель работы.

Изучить понятие дерева, леса и бинарного дерева, освоить программирование алгоритмов на бинарных деревьях. Реализовать бинарное дерево для решения поставленной задачи.

Задание.

Вариант 6

Задано бинарное дерево b типа BT с произвольным типом элементов. Используя очередь, напечатать все элементы дерева b по уровням: сначала из корня дерева, затем (слева направо) \sim из узлов, сыновних по отношению к корню, затем (также слева направо) \sim из узлов, сыновних по отношению к этим узлам, и т. д.

Выполнение работы.

Сначала был написан класс бинарного дерева *BTree* с использованием шаблонов. У данного класса есть поля: *size*, хранящее в себе размер массива, *treePtr* — умный указатель (*unique_ptr*) на массив, хранящий в себе представление бинарного дерева, *brk* — в зависимости от типа T , это будет или целое число или символ, он нужен для инициализации пустых ячеек массива, хранящего бинарное дерево.

Методы класса:

BTree(std::string& s) — конструктор класса, принимает на вход строку — скобочное представление бинарного дерева. Создает массив, хранящий данное дерево.

void stringToMassive(std::unique_ptr<T[]>& ptr, std::string& s, int start, int end, int ptrIdx) — метод, который инициализирует массив, посредством считывания строки, представляющей скобочную запись дерева.

void horizontalTraversal() — метод, организующий обход дерева в ширину, используя очередь.

void lkpTour(std::unique_ptr<T[]>& ptr, int m = 0) — метод, организующий ЛКП обход дерева

void insertData(T data) — метод, который осуществляет вставку элемента *data* в дерево.

void deleteData(T data) — метод, который осуществляет удаление элемента *data* из дерева.

T returnChild(int index) — метод, который переставляет родителя и сына. Он используется в методе *deleteData()*.

int getSize () — метод, который возвращает размер массива.

bool getError() — метод, который возвращает поле *err*.

std::unique_ptr<T[]>& getPtr() — метод, который возвращает указатель на массив.

T getBrk() — метод, который возвращает поле *brk*.

bool checker(std::string & s) — метод, который проверяет строку на валидность.

std::unique_ptr<T[]>& getPtr() — метод, возвращающий указатель на массив. Используется в других методах.

void printer() — метод, распечатывающий бинарное дерево. Используется для отладки.

Алгоритм работы программы: строка, поступающая на вход, поступает в метод *checker()*, в котором проходит разные проверки на валидность. После этого, данной строкой инициализируется объект класса *Btree*. Строка поступает в метод *stringToMassive()*, который разбивает данную строку на 3 части: первая часть, содержащая корень дерева, две другие — левый и правый сыновья корня. Далее программа рекурсивно вызывает себя для каждого потомка корня, передавая в аргументы индексы начала и конца каждого потомка в строке. Когда строка раздроблена до того момента, что индексы указывают на листья какого-то корня, то происходит их считывание и запись в массив. Таким рекурсивным образом происходит инициализация

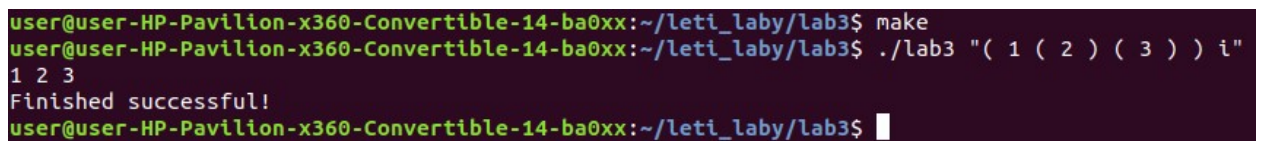
массива. В случае ошибки, программа поставит флажок *err*, который в дальнейшем будет проверяться при вызове других методов данного класса.

Далее программа обходит полученный массив в ширину: с помощью очереди получает элементы на всех уровнях и распечатывает элементы каждого уровня в одну строку.

Тестирование.

Запуск программы начинается с запуска команды *make* в терминале, что приведет к созданию исполняемого файла *lab3*. Запуск программы начинается с ввода команды *./lab3* в терминале в директории *lab3*. Тестирование же проводится с помощью скрипта *tester.py*, который запускается командой *python3 tester.py* в командной строке в директории *lab3*. В текстовых файлах лежат входные данные. Скрипт в результате тестирования выводит входные данные, и то, что вывела программа. Подавать на вход программе нужно последовательность символов и цифр в виде скобочной записи бинарного дерева (сначала идет корень, потом в скобках левый сын, после него в других скобках правый сын), символы должны быть разделены между собой пробелами, а также отделены пробелами от скобок. Также после того, как введена скобочная запись дерева, через пробел нужно ввести следующие символы: *c* — если данными бинарного дерева являются символы типа *char* и *i* — если данными бинарного дерева являются целые числа.

Далее приложен скриншот, который демонстрирует, как вводить данные вне модуля тестирования.



```
user@user-HP-Pavilion-x360-Convertible-14-ba0xx:~/leti_laby/lab3$ make
user@user-HP-Pavilion-x360-Convertible-14-ba0xx:~/leti_laby/lab3$ ./lab3 "( 1 ( 2 ) ( 3 ) ) i"
1 2 3
Finished successful!
user@user-HP-Pavilion-x360-Convertible-14-ba0xx:~/leti_laby/lab3$
```

Рисунок 1 — Запуск вне модуля тестирования

Результаты тестирования представлены в приложении Б.

Выводы.

Изучили понятие дерева, леса и бинарного дерева, освоили программирование алгоритмов на бинарных деревьях. Реализовали бинарное дерево для решения поставленной задачи. При написании программы использовали умные указатели и шаблоны.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <cstdlib>
#include <string>
#include <memory>
#include <limits.h>
#include <queue>

template<typename T>
class BTree{

    int size = 16;
    bool err = false;
    std::unique_ptr<T[]> treePtr;
    T brk;

public:

    BTree(std::string& s){
        treePtr = std::make_unique<T[]>(size);
        if (std::is_same<T, char>::value) {
            this->brk = CHAR_MIN;
        } else if (std::is_same<T, int>::value) {
            this->brk = INT_MIN;
        }
        for(int i = 0; i < size; i++){
            treePtr[i] = brk;
        }
        /*Считывание дерева в массив*/
        stringToMassive(treePtr, s, 0, s.length(), 0);
    }

    void stringToMassive(std::unique_ptr<T[]>& ptr, std::string&
s, int start, int end, int ptrIdx) {
        if(ptrIdx == this->size - 2 || ptrIdx == this->size - 1){
            int newSize = size*2;
            std::unique_ptr<T[]> newPtr =
std::make_unique<T[]>(newSize);
            for(int i = 0; i < newSize; i++){
                newPtr[i] = this->brk;
            }
            for(int i = 0; i < this->size; i++){
                newPtr[i] = ptr[i];
            }
            ptr = std::move(newPtr);
            this->size = newSize;
        }
        /*проверка на лист*/
        int brackets = 0;
        for (int i = start + 1; i < end; i++) {
            if (s[i] == '(') {
```

```

        brackets++;
    }
}
/*если лист - считать и положить в массив*/
if (!brackets) {
    std::string val = "";
    for (int i = start + 1; i < end; i++) {
        if(s[i] != ' ') {
            val += s[i];
        }
    }
    if (std::is_same<T, char>::value) {
        if(val != "") {
            ptr[ptrIdx] = val.c_str()[0];
        }
    } else if (std::is_same<T, int>::value) {
        if(val != "") {
            ptr[ptrIdx] = atoi(val.c_str());
        }
    }
    return;
} else {
    /*считываем корень*/
    std::string data = "";
    int leftLeft = 0;
    int rightLeft = 0;
    for (int i = start + 1; i < end; i++) {
        if (s[i] == '(') {
            leftLeft = i;
            break;
        }
        if(s[i] != ' ') {
            data += s[i];
        }
    }
    if (std::is_same<T, char>::value) {
        if(data != "") {
            ptr[ptrIdx] = data.c_str()[0];
        }
    } else if (std::is_same<T, int>::value) {
        if(data != "") {
            ptr[ptrIdx] = atoi(data.c_str());
        }
    }
    /*ищем границы левого поддерев*/
    int sum = 0;
    for (int i = leftLeft; i < end; i++) {
        if (s[i] == '(') {
            sum++;
        }
        if (s[i] == ')') {
            sum--;
        }
        if (!sum) {
            rightLeft = i;
            break;
        }
    }
}

```

```

    }
}
/*ищем границы правого поддерев*/
int j = 0;
while(s[rightLeft + j] != '('){
    if(rightLeft + j == end){
        err = 1;
        break;
        return;
    }
    j++;
}
int leftRight = rightLeft + j, rightRight = 0;
for (int i = leftRight; i < end; i++) {
    if (s[i] == '(') {
        sum++;
    }
    if (s[i] == ')') {
        sum--;
    }
    if (!sum) {
        rightRight = i;
        break;
    }
}
    stringToMassive(ptr, s, leftLeft, rightLeft, 2 *
ptrIdx + 1);
    stringToMassive(ptr, s, leftRight, rightRight, 2 *
ptrIdx + 2);
}
}

void horizontalTraversal(){
    std::queue<int> treeQueue;
    treeQueue.push(0);
    int it = 0, counter = 1;
    while(!treeQueue.empty()){
        it = treeQueue.front();
        treeQueue.pop();
        std::cout << treePtr[it] << ' ';
        if(treePtr[2*it + 1] != brk){
            treeQueue.push(2*it + 1);
        }
        if(treePtr[2*it + 2] != brk){
            treeQueue.push(2*it + 2);
        }
    }
    std::cout << '\n';
}

void printer(){
    for(int i = 0; i < this->size; i++){
        if(treePtr[i] != brk) {
            std::cout << treePtr[i];
        }else{
            std::cout << '#';
        }
    }
}

```



```

        }
    }
    std::cout << '\n';
}

void lkpTour(std::unique_ptr<T[]>& ptr, int m = 0){
    if(ptr[m] != brk){
        lkpTour(ptr, 2*m + 1);
        std::cout << ptr[m] << '\n';
        lkpTour(ptr, 2*m + 2);
    }
}

void insertData(T data){
    for(int i = 0; i < size; i++){
        if(treePtr[i] == brk){
            treePtr[i] = data;
            break;
        }
    }
}

void deleteData(T data){
    int delIndex = 0;
    for(int i = 0; i < size; i++){
        if(treePtr[i] == data){
            delIndex = i;
            break;
        }
    }

    if(treePtr[delIndex*2 + 1] == brk && treePtr[delIndex*2 +
2] == brk){
        treePtr[delIndex] = brk;
    }
    if(treePtr[delIndex*2 + 1] == brk && treePtr[delIndex*2 +
2] != brk){
        treePtr[delIndex] = treePtr[delIndex*2 + 2];
        treePtr[delIndex*2 + 2] = brk;
    }
    if(treePtr[delIndex*2 + 1] != brk && treePtr[delIndex*2 +
2] == brk){
        treePtr[delIndex] = treePtr[delIndex*2 + 1];
        treePtr[delIndex*2 + 1] = brk;
    }
    if(treePtr[delIndex*2 + 1] != brk && treePtr[delIndex*2 +
2] != brk){
        treePtr[delIndex] = returnChild(delIndex*2 + 1);
    }
}

T returnChild(int index){
    if(treePtr[index] == brk){
        return brk;
    }
    T data = treePtr[index];

```

```

        treePtr[index] = returnChild(2*index+ 1);

        return data;
    }

    BTree(const BTree<T>& tree):size(tree.size), err(tree.err),
    brk(tree.brk) {
        this->treePtr = std::make_unique<T[]>(tree.size);
        for(int i = 0; i < tree.size; i++){
            this->treePtr[i] = tree.treePtr[i];
        }
    }

    BTree<T>& operator= (BTree<T>& tree){
        if (this == &tree) {
            return *this;
        }
        this->err = tree.getError();

        for(int i = 0; i < tree.getSize(); i++){
            this->treePtr[i] = tree.getPtr()[i];
        }
        this->size = tree.getSize();
        this->brk = tree.getBrk();

        return *this;
    }

    int getSize () {
        return size;
    }

    bool getError() {
        return err;
    }

    std::unique_ptr<T[]>& getPtr() {
        return treePtr;
    }

    T getBrk() {
        return brk;
    }
};

bool checker(std::string & s){
    if(s[0] != '(' || s[s.length() - 1] != ')'){
        return false;
    }

    std::string inside = "";

    for(int i = 1; i < s.length() - 1; i++){

```

```

        if(s[i] != ' '){
            inside+=s[i];
        }
    }

    if(inside.empty()){
        return false;
    }

    int brackets = 0;
    for(int i = 0; i < s.length(); i++){
        if(s[i] == '('){
            brackets++;
        }
        if(s[i] == ')'){
            brackets--;
            if(brackets < 0){
                return false;
            }
        }
    }
    if(brackets){
        return false;
    }

    std::string father = "";
    for(int i = 1; i < s.length(); i++){
        if(s[i] == '('){
            break;
        }
        if(s[i] != ' '){
            father += s[i];
        }
    }

    if(father.empty()){
        return false;
    }

    std::string brs = "";
    int i = 0;
    while(i < s.length()){
        if(s[i] == '('){
            brs += s[i];
        }
        if(s[i] == ')'){
            if(brs.empty()){
                return false;
            }
            else {
                int m = brs.length();
                brs.erase(m - 1, 1);
            }
        }
        i++;
    }
    return true;

```

```

}

int main(int argc, char* argv[]){

    if(argc < 2){
        std::cout << "Incorrect string!" << '\n';
        return 0;
    }

    std::string s(argv[1]);

    std::string type = "";
    type += s[s.length() - 1];
    s.erase(s.length() - 2);

    if(type != "c" && type != "i") {
        std::cout << "Incorrect string!" << '\n';
        return 0;
    }

    if(!checker(s)){
        std::cout << "Incorrect string!" << '\n';
        return 0;
    }

    if(type == "c"){
        BTree<char> BinaryTree(s);
        if(BinaryTree.getError()){
            std::cout << "Incorrect string!" << '\n';
            return 0;
        }else{
            //BinaryTree.printer();
            BinaryTree.horizontalTraversal();
        }
    }

    }else if (type == "i"){
        BTree<int> BinaryTree(s);
        if(BinaryTree.getError()){
            std::cout << "Incorrect string!" << '\n';
            return 0;
        }else{
            //BinaryTree.printer();
            BinaryTree.horizontalTraversal();
        }
    }

    std::cout << "Finished successful!" << '\n';
    return 0;
}

```

ПРИЛОЖЕНИЕ Б

ТЕСТИРОВАНИЕ

Таблица Б.1 - Примеры тестовых случаев

№ п/п	Входные данные	Выходные данные	Комментарии
1.	(a (b)) ()) (c (d) ())) c	Incorrect string!	Неправильный порядок скобок
2.	(a (b) (c)) c	a b c Finished successful!	
3.	() c	Incorrect string!	Пустое дерево,
4.	(a (b () ()) (c (e) ())) c	a b c e Finished successful!	
5.	(1 (2) (3)) i	1 2 3 Finished successful!	
6.	agjps i	Incorrect string!	Введенная строка не является скобочной записью дерева
7.		Incorrect string!	Введена пустая строка
8.	(a (b (d) (e)) (f (g) (h))) c	a b f d e g h Finished successful!	
9.	(a b () ()) c (e) ()) c	Incorrect string!	Пропущены скобки
10.	(1000 (20 () ()) (30 (6) ())) i	1000 20 30 6 Finished successful!	