

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: Бинарные деревья

Студентка гр. 9304

Рослова Л.С

Преподаватель

Филатов А.Ю.

Санкт-Петербург

2020

Цель работы.

Изучить понятие бинарного дерева. Реализовывать программу с использованием бинарного дерева в языке C++.

Задание.

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

- напечатать элементы из всех листьев дерева *b*;
- подсчитать число узлов на заданном уровне *n* дерева *b* (корень считать узлом 1-го уровня).

Выполнение работы.

Программа принимает на вход текстовый файл, в которой две строки: первая задает структуру бинарного дерева, а вторая определяет уровень поиска узлов. Программа производит считывание данных строк, подает первую в качестве аргумента в конструктор объекта класса *Tree*. Конструктор производит проверку методом *checkStruct*, если строка удовлетворяет требованиям — на ее основе метод *createBT* рекурсивно создает массив указателей на элементы класса *Node*, в которых хранятся данные шаблонного типа *base* (данный тип позволяет использовать только *char*), в противном случае — вектор остается пустым. Дабы избежать излишней резервации памяти, изначально в массиве присутствует 64 доступных элемента, если данного значения не хватит — программа выделит дополнительные блоки того же размера. В классе *Tree* так же реализованы методы взаимодействия с бинарным деревом: *findLeafs* выводит все листья и их позиции в массиве, *nodesOnLevel* выводит узлы на заданном уровне, *addNode* добавляет новый элемент в массив. Новый элемент занимает первую свободную ячейку, дабы нивелировать расстояние между элементами. В дополнение были реализованы следующие методы: *removeNode* удаляет узел по его индексу в массиве, *lkr* совершает обход дерева в обратном порядке, выводя по пути узлы.

Разработанный программный код см. в приложении А.

Формат входных и выходных данных.

На вход программе подается текстовый файл со структурой бинарного дерева.

Программа должна вывести все листья, а так же узлы на заданном уровне.

Тестирование.

Для проведения тестирования был написан bash-скрипт `./script`. Скрипт запускает программу где в качестве входных аргументов служат заранее подготовленные файлы, расположенные в папке `./Tests`

Результаты тестирования см. в приложении Б.

Выводы.

Было изучено понятие бинарного дерева. Было реализовано бинарное дерево с помощью языка программирования C++, с использованием шаблонов.

Была реализована программа, создающая бинарное дерево на основе массива. Использование данной программы будет эффективно, если структура представляет из себя полное дерево.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <fstream>
#include <memory>

#include "Node.h"
#include "Tree.h"

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

    setlocale(LC_ALL, "ru");

    if(argc != 2){
        std::cout << "Неверное кол-во аргументов!" << std::endl;
        return -1;
    }

    std::fstream file(argv[1]);

    if(!file){
        std::cout << "Файл " << argv[0] << " не может быть открыт
на чтение!" << std::endl;
        return -1;
    }

    std::string structTree {};
    std::string levelFind {};
    getline(file, structTree);
    getline(file, levelFind);

    file.close();

    std::unique_ptr<Tree<char>> myTree(new Tree<char>(structTree));

    if(myTree->getArr().size() == 0){
        std::cout << "Неверная структура дерева!" << std::endl;
        return -1;
    }

    //myTree->removeNode(0);
    //myTree->addNode('p');
    myTree->findLeafs();
    myTree->nodesOnLevel(levelFind);
    //myTree->lkp();

    return 0;
}
```

Название файла: Tree.h

```
#pragma once

#include <string>
#include <vector>
#include "Node.h"

#define ARR_SIZE 64 // Не бейте, удобно задавать выделяемые блоки памяти

template<typename base>
class Tree{
    std::vector<Node<base>*> vec;
    bool checkStruct(std::string structTree);
    void createBT(std::string structTree);
public:
    Tree(std::string structTree);
    Tree(Tree<base>&& tree);
    ~Tree();
    Tree<base>& operator=(Tree<base>&& tree);
    std::vector<Node<base>*> getArr();
    void findLeafs();
    void nodesOnLevel(std::string levelFind);
    void addNode(base newData);
    void removeNode(size_t index);
    void lkp();
};

template<typename base>
Tree<base>::Tree(std::string structTree){
    if(!checkStruct(structTree)){
        vec.clear();
    }else{
        vec.resize(ARR_SIZE);
        createBT(structTree);
    }
}

template<typename base>
Tree<base>::Tree(Tree<base>&& tree){
    std::swap(tree->vec, vec);
}

template<typename base>
Tree<base>& Tree<base>::operator=(Tree<base>&& tree){
    vec = std::move(tree->vec);
}

template<typename base>
```

```

Tree<base>::~~Tree(){
    for(size_t i = 0; i < vec.size(); i++){
        if(vec[i]){
            delete vec[i];
        }
    }
}

template<typename base>
bool Tree<base>::checkStruct(std::string structTree){
    int level = 0;
    int node = 0;

    if(structTree.size() == 0){
        return 0;
    }

    for(size_t i = 0; i < structTree.size(); i++){
        if(isalpha(structTree[i])){
            node++;
        }

        if(structTree[i] == '('){
            level++;
        }

        if(structTree[i] == ')'){
            level--;
            node--;
        }

        if((level - node) < 0){
            return 0;
        }

        if((node - level < -1)){
            return 0;
        }

        if(level < 0){
            return 0;
        }
    }

    if(level != 0){
        return 0;
    }

    if(structTree[0] != '('){
        return 0;
    }
}

```

```

    if(structTree[structTree.size() - 1] != ' '){
        return 0;
    }

    return 1;
}

template<typename base>
void Tree<base>::createBT(std::string structTree){

    size_t first = 0;
    size_t second = 0;
    size_t count = 0;
    size_t arrLevel = 1;
    std::vector<Node<base>*> nodePtr = vec;

    auto PR = [&second, &nodePtr, &structTree, &arrLevel](size_t count,
size_t first, auto &&PR){

        second++;

        while(count > nodePtr.size() - 1){
            std::vector<Node<base>*> add{};
            arrLevel++;
            add.resize(ARR_SIZE * arrLevel);
            for(size_t i = 0; i < nodePtr.size(); i++){
                if(nodePtr[i] != nullptr){
                    add[i] = nodePtr[i];
                }
            }
            nodePtr = add;
        }

        if(structTree[first] == '(' && structTree[second] == ' '){
            return;
        }

        if(structTree[first] == '(' && structTree[second] == '('){

            if(second - first == 2){
                PR((count - 1) * 2 + 1, second, PR);
            }else{
                PR((count - 1) * 2 + 2, second, PR);
            }
        }

        if(structTree[first] == '(' && isalpha(structTree[second])){

            if(!first){
                nodePtr[0] = new Node<base>(structTree[second]);
            }else{
                nodePtr[count] = new Node<base>(structTree[second]);
            }
        }
    };
}

```

```

        count++;
    }

    PR(count, first, PR);
};

PR(count, first, PR);
vec = nodePtr;
}

template<typename base>
std::vector<Node<base>*> Tree<base>::getArr(){
    return vec;
}

template<typename base>
void Tree<base>::findLeafs(){
    for(size_t i = 0; i < vec.size(); i++){
        if(vec[i]){
            if(vec[i * 2 + 1] == nullptr && vec[i * 2 + 2] == nullptr){
                std::cout << vec[i]->getData() << " - Лист на позиции "
<< i << '\n';
            }
        }
    }
}

template<typename base>
void Tree<base>::nodesOnLevel(std::string levelFind){
    size_t level = atoi(levelFind.c_str());

    size_t node1 = 1;
    bool flag = 0;

    if(level > 0){
        for(size_t i = 0; i < level - 1; i++){
            node1 = node1 * 2;
        }

        node1--;

        std::cout << "На уровне " << level << " находятся следующие
узлы:\n";

        for(size_t j = 0; j < node1 + 1; j++){
            if(vec[(node1 + j) * 2 + 1] != nullptr){
                flag = 1;
            }
        }
    }
}

```



```

        if(vec[(node1 + j) * 2 + 2] != nullptr){
            flag = 1;
        }

        if(flag){

            std::cout << vec[node1 + j]->getData() << " ";

        }

        flag = 0;

        std::cout << std::endl;

    }else{
        std::cout << "Некорректный уровень!\n";
    }
}

template<typename base>
void Tree<base>::addNode(base newData){
    for(size_t i = 0; i < vec.size(); i++){
        if(vec[i] == nullptr){
            vec[i] = new Node(newData);
            break;
        }
    }
}

template<typename base>
void Tree<base>::removeNode(size_t index){

    if(index > vec.size()){
        std::cout << "Индекс удаления выходит за пределы массива!" <<
std::endl;
    }else{
        if(vec[index]){
            base newData;
            for(size_t i = vec.size() - 1; i >= 0; i--){
                if(vec[i]){
                    newData = vec[i]->getData();
                    delete vec[i];
                    vec[i] = nullptr;
                    break;
                }
            }
            vec[index]->setData(newData);
        }else{
            std::cout << "Элемент на заданном индексе отсутствует!" <<
std::endl;
        }
    }
}

```

```
}
```

```
template<typename base>
void Tree<base>::lkp(){
    std::vector<Node<base>*> arrNode = vec;
    size_t index = 0;

    auto LKP = [&arrNode](size_t index, auto &&LKP){

        if(!arrNode[index]){
            return;
        }else{
            LKP(index * 2 + 1, LKP);
            std::cout << arrNode[index]->getData() << ' ';
            LKP(index * 2 + 2, LKP);
        }
    };

    LKP(index, LKP);
    std::cout << '\n';
}
```

Название файла: Node.h

```
#pragma once
```

```
template<typename base>
class Node{
    base data;
public:
    Node(base newData);
    base getData();
    void setData(base newData);
};
```

```
template<typename base>
Node<base>::Node(base newData){
    data = newData;
}
```

```
template<typename base>
base Node<base>::getData(){
    return data;
}
```

```
template<typename base>
void Node<base>::setData(base newData){
    data = newData;
}
```

Название файла: script

```
#!/bin/bash
```

```
arg1=$(cat Tests/test1.txt)
echo -e "_____ \n"
echo -e "Test 1:\n"
echo "argument 1 = $arg1"
echo
./lab3 ./Tests/test1.txt
```

```
arg1=$(cat Tests/test2.txt)
echo -e "_____ \n"
echo -e "Test 2:\n"
echo "argument 1 = $arg1"
echo
./lab3 ./Tests/test2.txt
```

```
arg1=$(cat Tests/test3.txt)
echo -e "_____ \n"
echo -e "Test 3:\n"
echo "argument 1 = $arg1"
echo
./lab3 ./Tests/test3.txt
```

```
arg1=$(cat Tests/test4.txt)
echo -e "_____ \n"
echo -e "Test 4:\n"
echo "argument 1 = $arg1"
echo
./lab3 ./Tests/test4.txt
```

```
arg1=$(cat Tests/test5.txt)
echo -e "_____ \n"
echo -e "Test 5:\n"
echo "argument 1 = $arg1"
echo
./lab3 ./Tests/test5.txt
echo -e "_____ \n"
```

ПРИЛОЖЕНИЕ Б

ТЕСТИРОВАНИЕ ПРОГРАММЫ

Результаты тестирования представлены в табл.1

Таблица 1 – Результаты тестирования

№	Входные данные	Выходные данные	Комментарии
1.	(a(b(d)(e))(c(f)(g(z)(x(q)(w(e)(r(t)(y))))))))	d - Лист на позиции 3 e - Лист на позиции 4 f - Лист на позиции 5 z - Лист на позиции 13 q - Лист на позиции 29 e - Лист на позиции 61 t - Лист на позиции 125 y - Лист на позиции 126 На уровне 3 находятся следующие узлы: g	Произошло расширение памяти на 1 дополнительный блок
2.	(a(b(d(h)(i))(e(k)(l)))(c(f(m)(n))(g(o)(p))))	h - Лист на позиции 7 i - Лист на позиции 8 k - Лист на позиции 9 l - Лист на позиции 10 m - Лист на позиции 11 n - Лист на позиции 12 o - Лист на позиции 13 p - Лист на позиции 14 На уровне 3 находятся следующие узлы: d e f g	Выводится лист, его позиция, а также узлы на соответствующем уровне
3.)a(b)(c)(Неверная структура дерева!	Кривые скобки
4.	(aa(b)(c))	Неверная структура дерева!	Разрешен только один символ.
5.	((O))	Неверная структура дерева!	Отсутствуют данные для узлов.

