

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

Кузнецов Р.В.

Преподаватель

Филатов А.Ю.

Санкт-Петербург

2020

Цель работы.

Научиться работать с деревьями, используя язык C++.

Задание.

18У. Бинарное дерево называется бинарным деревом поиска, если для каждого его узла справедливо: все элементы правого поддерева больше этого узла, а все элементы левого поддерева – меньше этого узла.

Бинарное дерево называется пирамидой, если для каждого его узла справедливо: значения всех потомков этого узла не больше, чем значение узла.

Для заданного бинарного дерева с числовым типом элементов определить, является ли оно бинарным деревом поиска и является ли оно пирамидой.

Основные теоретические положения.

Дерево – конечное множество T , состоящее из одного или более узлов, таких, что

а) имеется один специально обозначенный узел, называемый корнем данного дерева;

б) остальные узлы (исключая корень) содержатся в $m \geq 0$ попарно не пересекающихся множествах T_1, T_2, \dots, T_m , каждое из которых, в свою очередь, является деревом. Деревья T_1, T_2, \dots, T_m называются поддеревьями данного дерева.

Каждый узел дерева является корнем некоторого поддерева. В том случае, когда множество поддеревьев такого корня пусто, этот узел называется концевым узлом, или листом. Уровень узла определяется рекурсивно следующим образом:

- 1) корень имеет уровень 1;
- 2) другие узлы имеют уровень, на единицу больший их уровня в содержащем их поддереве этого корня.

Бинарное дерево – конечное множество узлов, которое либо пусто, либо состоит из корня и двух непересекающихся бинарных деревьев, называемых правым поддеревом и левым поддеревом.

Объяснение работы алгоритма.

На вход программа запрашивает у пользователя бинарное дерево в виде упрощенной скобочной записи, где пустое дерево обозначается с помощью '!'. Результатом работы программы являются две строки, сообщающие, является ли данное дерево бинарным деревом поиска или пирамидой.

Функция `getInt()` принимает ссылку на итератор и итератор на конец строки, из которой создается дерево, и возвращает считанное целочисленное значение, перемещая итератор на первый символ, не относящийся к считанному числу.

Node – структурная единица бинарного дерева. Она имеет поле данных и два указателя: на левое и правое поддерево. Структура Node представлена на рисунке 1.



Рисунок 1 – структура Node

BinTree – класс бинарного дерева. Он имеет поле указателя на `NodePtr` и набор методов для работы с ним:

Конструктор класса `BinTree` из итераторов на начало и конец строки содержит рекурсивная лямбда функцию `makeBinTree()`, которая принимает ссылку на указатель на ноду, в которую требуется записать дерево, а также захватывает по ссылке итераторы на начало и конец строки, из которой нужно его создать дерево. На рисунке 2 представлен пример созданного дерева из строки `(0(1(3!(7))(4))(2(5(8)(9))(6!(10(11))))))`.

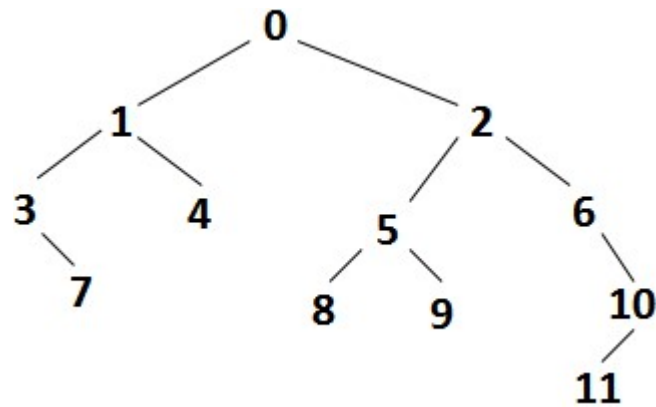


Рисунок 2 – пример созданного бинарного дерева

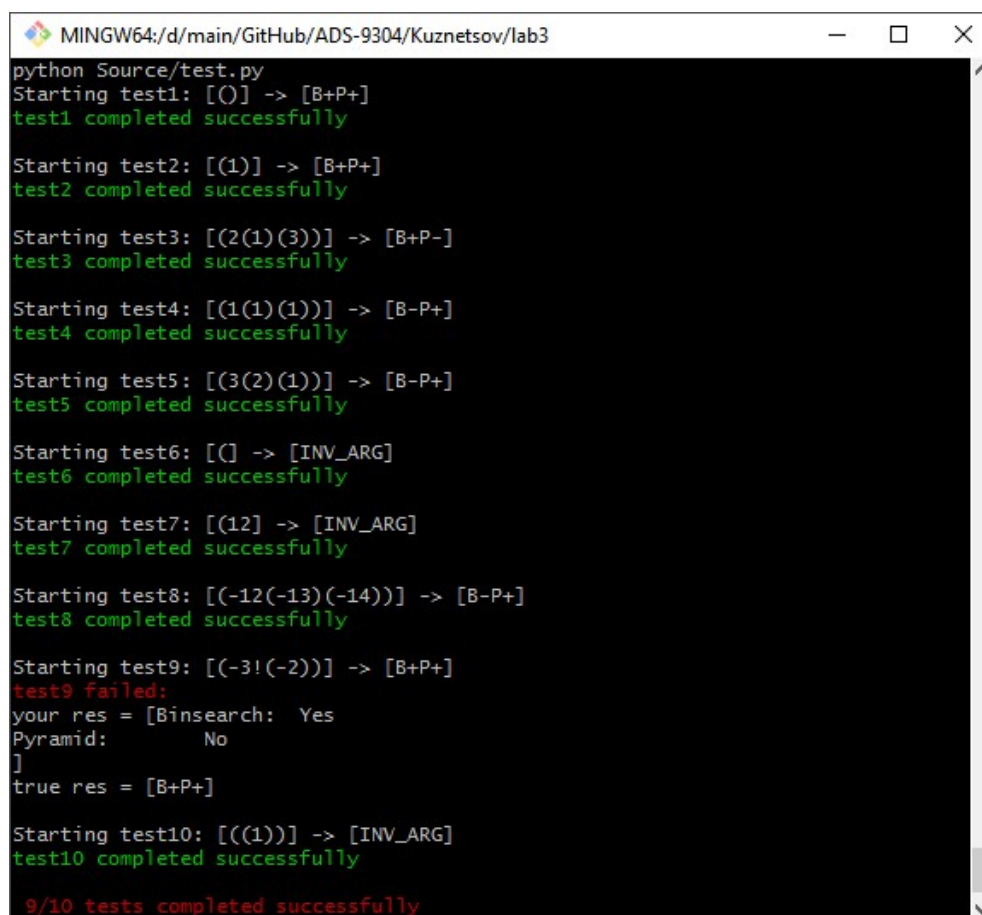
Метод `BinSearchCheck()` внутри содержит рекурсивную лямбда функцию `binCheck()`, реализующую основную часть работы. `binCheck()` проходит по дереву алгоритмом DFS с порядком ЛКП и проверяет, чтобы все данные в порядке обхода шли по возрастанию. Если условие соблюдается – `BinSearchCheck()` возвращает `true`, иначе – `false`.

Метод `PyramideCheck()` возвращает значение описанной внутри лямбда функции `_pyramideCheck()`, которая принимает по ссылке константный указатель на дерево и возвращает логическое значение. Она рекурсивно проверяет, чтобы значение всех потомков каждой ноды было не больше значения данной ноды. Если условие соблюдается – `_pyramideCheck()` возвращает `true`, иначе – `false`.

Также были перегружены копирующий конструктор и оператор копирующего присваивания. Их функционал описан рекурсивной приватной функцией `_сору()`, принимающей ссылку на указатель на ноду, в которую требуется скопировать дерево и константную ссылку на указатель на ноду, из которой требуется скопировать дерево. Она, при условии существования узла дерева донора, создает соответствующий узел в дереве реципиенте с эквивалентным полем `data` и запускает эту же функцию для левых и правых потомков дерева.

Тестирование.

Для тестирования был написан скрипт на Python и Makefile, упрощающий с ним взаимодействие. Тестирующая программа проверяет результат работы основной программы и выводит несоответствия, если они найдены. Пример работы Python скрипта в случае нахождения несоответствия представлен на рисунке 3. Результаты тестирования представлены в таблице Б.1 и продолжаются в таблице Б.2 в приложении. Выходные данные в таблицах сокращены: так B+P- означает, что дерево является бинарным деревом поиска, но не является пирамидой, а INV_ARG означает инвалидность аргумента.



```
python Source/test.py
Starting test1: [()] -> [B+P+]
test1 completed successfully

Starting test2: [(1)] -> [B+P+]
test2 completed successfully

Starting test3: [(2(1)(3))] -> [B+P-]
test3 completed successfully

Starting test4: [(1(1)(1))] -> [B-P+]
test4 completed successfully

Starting test5: [(3(2)(1))] -> [B-P+]
test5 completed successfully

Starting test6: [()] -> [INV_ARG]
test6 completed successfully

Starting test7: [(12)] -> [INV_ARG]
test7 completed successfully

Starting test8: [(-12(-13)(-14))] -> [B-P+]
test8 completed successfully

Starting test9: [(-3!(-2))] -> [B+P+]
test9 failed:
your res = [Binsearch: Yes
Pyramid:      No
]
true res = [B+P+]

Starting test10: [((1))] -> [INV_ARG]
test10 completed successfully

9/10 tests completed successfully
```

Рисунок 3 – Пример работы тест скрипта в случае несоответствия

Таблица Б.1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	()	V+P+	Пустое дерево
2.	(1)	V+P+	Дерево из одного элемента
3.	(2(1)(3))	V+P-	Дерево поиска

Выводы.

В процессе выполнения работы было проведено ознакомление с бинарными деревьями, алгоритмами его обхода.

Была реализована программа, определяющая является ли бинарное дерево пирамидой или бинарным деревом поиска. Также была реализована тестирующая программа на языке Python, был написан Makefile, упрощающий взаимодействие с программой.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <memory>
#include <string>
#include <iterator>
#include <queue>
#include <algorithm>

#define invStringArg() {\
    std::cout << "ERROR: given string tree is invalid";\
    exit(EXIT_SUCCESS);\
}

int getInt(std::string::iterator& it, const std::string::iterator
end) {
    int val = 0;
    bool isNegative = false;
    if (*it == '-') {
        isNegative = true;
        it++;
    }
    if (!isdigit(*it) && *it == '(') invStringArg();
    while (it != end && isdigit(*it))
        val = val * 10 + *(it++) - '0';
    if (it == end) invStringArg();
    return isNegative ? -val : val;
}

template <typename T>
struct Node {
    int data = 0;
    std::shared_ptr<Node> left, right;
    Node(T data) : data(data) {}
};

template <typename T>
using NodePtr = std::shared_ptr<Node<T>>;
template <typename T>
class BinTree {
public:
    BinTree() : head(nullptr) {}
    BinTree(const std::string::iterator begin, const
std::string::iterator end) {
        auto it = begin;
        auto makeBinTree = [&it, &end](NodePtr<int> & node, auto
makeBinTree) -> void {
            if (*it == ')') return;
            if (*it == '!') {
                it++;
                return;
            }
            if (*(it++) != '(' || it == end) invStringArg();
            node = std::make_shared<Node<int>>(getInt(it, end));
            makeBinTree(node->left, makeBinTree);
            makeBinTree(node->right, makeBinTree);
            it++;
        };
    }
};
```

```

        };
        makeBinTree(head, makeBinTree);
    }
    BinTree& operator=(const BinTree<T> & val) {
        _copy(head, val.head);
        return *this;
    }
    BinTree(const BinTree<T>& val) {
        _copy(head, val.head);
    }
    bool BinSearchCheck() {
        int min = INT_MIN;
        bool isBinSearch = true;
        auto binCheck = [&isBinSearch, &min](const NodePtr<T> &
node, auto && binCheck) {
            if (!node || !isBinSearch)
                return;
            binCheck(node->left, binCheck);
            isBinSearch &= node->data > min;
            min = node->data;
            binCheck(node->right, binCheck);
        };
        binCheck(head, binCheck);
        return isBinSearch;
    }
    bool PyramideCheck() {
        auto _pyramideCheck = [] (const NodePtr<T> & node, auto &&
_pyramideCheck) -> bool {
            return ((node->left) ? (node->data >=
node->left->data) && _pyramideCheck(node->left, _pyramideCheck) : true) \
                && ((node->right) ? (node->data >=
node->right->data) && _pyramideCheck(node->right, _pyramideCheck) : true);
        };
        return _pyramideCheck(head, _pyramideCheck);
    }
private:
    NodePtr<T> head;
    void _copy(NodePtr<T> & dest, const NodePtr<T> & source){
        if (source) {
            dest = std::make_shared<Node<T>>(source->data);
            _copy(dest->left, source->left);
            _copy(dest->right, source->right);
        }
    };
};
int main() {
    std::string input;
    std::getline(std::cin, input);
    BinTree<int> tree(input.begin(), input.end());
    std::cout << "Binsearch:\t" << (tree.BinSearchCheck() ? "Yes" :
"No") \
        << "\nPyramid:\t" << (tree.PyramideCheck() ? "Yes" : "No")
<< '\n';
    return 0;
}

```


ПРИЛОЖЕНИЕ Б

ТЕСТИРОВАНИЕ

Если результаты тестирования велики (больше 1 страницы), то их выносят в приложение.

Процесс тестирования можно представить в виде таблицы, например:

Таблица Б.2 - Примеры тестовых случаев

№ п/п	Входные данные	Выходные данные	Комментарии
4.	(1(1)(1))	B-P+	Дерево одинаковых элементов
5.	(3(2)(1))	B-P+	Пирамида
6.	(INV_ARG	Строка не является деревом
7.	(12	INV_ARG	У открывающейся скобки нет пары
8.	(-12(-13)(-14))	B-P+	Пирамида из отрицательных элементов
9.	(-3!(-2))	B+P-	Отсутствующее левое поддерев
10.	((1))	INV_ARG	Лишние скобки