**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: Бинарные деревья

|  |  |  |
| --- | --- | --- |
| Студент гр. 8304 |  | Алтухов А.Д. |
| Преподаватель |  | Фирсов М.А. |

Санкт-Петербург

2019

**Цель работы.**

Ознакомиться с линейными структурами данных, обеспечивающими доступ к данным только через начало и/или конец структуры, способами их реализации.

**Задание.**

Вариант 2-в.

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

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

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

- напечатать элементы из всех листьев дерева *b*;

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

**Описание алгоритма.**

Определение максимальной глубины: рекурсивно вызывается функция подсчета глубины для левого и правого поддеревьев, на каждом уровне к текущей высоте добавляется единица. Учитывается только один путь до листа — максимальный.

Вычисление длины внутреннего пути: если существуют поддеревья, то к счетчику добавляется числовое значение уровня дерева, который сейчас обрабатывается алгоритмом. После этого функция рекурсивно вызывается для поддерева с увеличением уровня.

Печать элементов из листьев: если поддеревьев нет, то печатаем элемент. В обратном случае рекурсивно вызываем функцию для существующих поддеревьев.

Подсчет узлов на заданном уровне дерева: в случае, если уровень узла соответствует искомому, возвращаем единицу, иначе рекурсивно вызываем функцию для поддеревьев и складываем возвращенные результаты.

**Основные функции и структуры.**

class **binaryTree** — класс, содержащий в себе информацию о бинарном дереве. Присутствуют следующие поля:

Elem<T>\* arr — содержит все элементы дерева.

int curIndex — номер обрабатываемого элемент.

int\* infoArr — информация о дереве — занимаемая память, следующая свободная ячейка.

bool isMainTree — является ли объект основным деревом или поддеревом.

struct **Elem** — структура, содержащая элемент дерева.

T value — значение узла.

int left — индекс левого поддерева в массиве arr.

int right — индекс правого поддерева в массиве arr.

Существуют следующие методы для взаимодействия с бинарным деревом:

bool **isEmptyTree**() const;

bool **isEmptyElem**() const — проверяет, существует ли дерево/элемент.

T **getRoot**() const — возвращает значение текущего узла.

binaryTree<T> **getLeftTree**();

binaryTree<T> **getRightTree**() — возвращает левое/правое поддерево.

void **setLeft**(T newValue);

void **setRight**(T newValue);

void **setRoot**(T newValue) — устанавливает новое значение в узел слева/справа/текущий узел.

bool **readTree**(std::ifstream& inputStream) — преобразует строковое представление дерева в необходимую для дальнейшей обработки форму.

int **calcHeight**();

int **pathLength**(int level=1);

void **printLeaves**();

int **countNodesOnLevel**(int level) — функции, выполняющие соответствующие заданию подзадачи. Их описание расположено выше.

**Тестирование.**

Некорректные данные:

|  |  |  |  |
| --- | --- | --- | --- |
| № | Ввод | Ожидаемый ответ | Полученный ответ |
| 1 | 12345 | Сообщение об ошибке | Нет открывающей скобки |
| 2 | (1# | Сообщение об ошибке | Лишний # |
| 3 | 2) | Сообщение об ошибке | Нет открывающей скобки |

Корректные данные:

|  |  |  |  |
| --- | --- | --- | --- |
| № | Ввод | Ожидаемый ответ | Полученный ответ |
| 4 | (1) | Высота дерева: 1  Длина внутреннего пути: 0  Список листьев: 1  Узлов на уровне 3: 0 | Высота дерева: 1  Длина внутреннего пути: 0  Список листьев: 1  Узлов на уровне 3: 0 |
| 5 | (1 (2)(3) ) | Высота дерева: 2  Длина внутреннего пути: 2  Список листьев: 2 3  Узлов на уровне 3: 0 | Высота дерева: 2  Длина внутреннего пути: 2  Список листьев: 2 3  Узлов на уровне 3: 0 |
| 6 | (1 (2 (3 (4)(5) ) ) ) | Высота дерева: 4  Длина внутреннего пути: 9  Список листьев: 4 5  Узлов на уровне 3: 1 | Высота дерева: 4  Длина внутреннего пути: 9  Список листьев: 4 5  Узлов на уровне 3: 1 |
| 7 | (1 (2 (3 (5))(4))(6)) | Высота дерева: 4  Длина внутреннего пути: 9  Список листьев: 5 4 6  Узлов на уровне 3: 2 | Высота дерева: 4  Длина внутреннего пути: 9  Список листьев: 5 4 6  Узлов на уровне 3: 2 |
| 8 | (1 (2 (3 (5))(4))(6 (7) (8))) | Высота дерева: 4  Длина внутреннего пути: 13  Список листьев: 5 4 7 8  Узлов на уровне 3: 4 | Высота дерева: 4  Длина внутреннего пути: 13  Список листьев: 5 4 7 8  Узлов на уровне 3: 4 |
| 9 | (1 (2 (3 (5))(4))(6 #(8))) | Высота дерева: 4  Длина внутреннего пути: 11  Список листьев: 5 4 8  Узлов на уровне 3: 3 | Высота дерева: 4  Длина внутреннего пути: 11  Список листьев: 5 4 8  Узлов на уровне 3: 3 |

**Выводы.**

В ходе выполнения лабораторной работы была исследована такая структура данных, как бинарное дерево, изучены методы работы с ней. Были реализованы некоторые алгоритмы для взаимодействия с деревом.