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

КУРСОВАЯ РАБОТА (КУРСОВОЙ ПРОЕКТ)

по дисциплине «Алгоритмы и Структуры данных»

Тема: АВЛ-дерево (вставка и поиск).

Студент гр. 0381 _____ Дзаппала Д.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург

2021

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ (КУРСОВОЙ ПРОЕКТ)

Студент Дзаппала Д.

Группа 0381

Тема работы: АВЛ-деревья — вставка и поиск.

Содержание пояснительной записки:

- Содержание
- Введение
- Выполнение работы
- Список используемых источников

Предполагаемый объем пояснительной записки:

Не менее 00 страниц.

Дата выдачи задания: 18.10.2021

Дата сдачи реферата: 22.12.2021

Дата защиты реферата: 23.12.2021

Студент _____ Дзаппала Д.

Преподаватель _____ Берленко Т.А.

АННОТАЦИЯ

Курсовая работа представляет из себя программу, взаимодействие с которой происходит через консоль. В программе происходит демонстрация операций вставки и поиска в АВЛ-дереве. Программу можно запустить как без входных данных, так и с входными в консоли. После чего, пользователю будет предложено вводить команды в консоль в режиме runtime: «insert» и «find». После команд требуется писать число, которое пользователь хочет вставить (или найти) в дерево.

SUMMARY

Course work is a program that interacts with through the console. The program demonstrates insertion and search operations in the AVL tree. The program can be run both without input data, and with input in the console. After that, the user will be prompted to enter commands into the console: "insert" and "find". After the commands, you need to write the number that the user wants to insert (or find) into the tree.

СОДЕРЖАНИЕ

1. ВВЕДЕНИЕ

Цель работы:

Продемонстрировать работу структуру данных АВЛ-дерево, а также как работают алгоритмы вставки и поиска в этой СД.

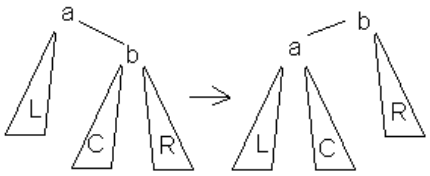
Для выполнения поставленной задачи требуется изучить теоретический материал по данной СД, реализовать саму структуру, протестировать программу и убедиться, что все работает правильно.

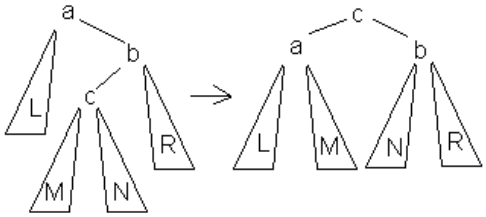
2. ТЕОРИЯ

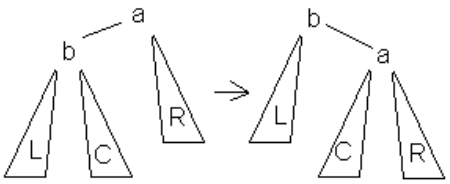
Что есть AVL-дерево? AVL-дерево (англ. *AVL-Tree*) — сбалансированное двоичное дерево поиска, в котором поддерживается следующее свойство: для каждой его вершины высота её двух поддеревьев различается не более чем на 1.

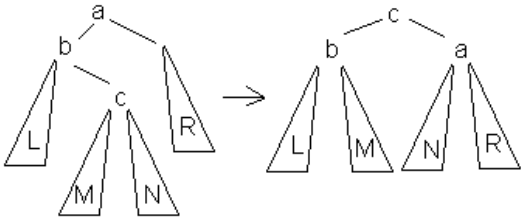
Скорость работы операций (поиск, вставка, удаление) работают за O(log(n)). Такая сложность достигается благодаря основному свойству АВЛ-дерева. Балансировка гарантирует, что не будет хулшего случая (как в бинарном дереве поиска) , когда вставка будет O(n) (все сыновья — правые, и получается (примерно) список).

Балансировкой вершины называется операция, которая в случае разницы высот левого и правого поддеревьев = 2, изменяет связи предок-потомок в поддереве данной вершины так, что разница становится <= 1, иначе ничего не меняет. Указанный результат получается вращениями поддерева данной вершины. Используются 4 типа вращения: малое левое вращение, большое левое вращение, малое правое вращение, большое правое вращение.









Алгоритм вставки вершин: Вставка, по большому счету, выполняется также, как и в простых бин. деревьях поиска. Идем вниз по дереву, спускаясь в правое или левое поддерево, в зависимости от результата сравнения ключа в текущем узле дерева и вставляемого ключа. Отличие заключается в том, что после возвращения из рекурсии (после вставки), осуществляется балансировка текущего узла.

Алгоритм поиска ключа: также как и в алгоритме вставки, проходимся вниз по дереву, идя в правое поддерево или левое, в зависимости от значений ключа в текущем узле и вставляемым. Рекурсивно идем, пока, либо не найдем элемент или пока указатель, передаваемый в функцию, не будет nullptr. После чего возвращаем указатель, проверяем nullptr он или нет.

3. ВЫПОЛНЕНИЕ РАБОТЫ

АВЛ-дерево реализовано в классе AVL_Tree, и содержит в себе только одно поле — root, указатель на корень. Тип узлов — node. node является структурой, обладающей полями key(int), height(uint8_t), left и right(node*), inserted и catched(bool). Последние два поля нужны для вывода информации в консоль. Left и right — указатели на левого и правого потомка. Переменная key — ключ, по которому идем сравнение при обходе дерева. Поле height — высоту поддерева с корнем в данном узле. Далее, для удобства, были написаны вспомогательные функции:

- getHeight(node*) - возвращает высоту поддерева с корнем в переданном узле.
- balanceFactor(node*) - возвращает разницу высот правого и левого поддеревьев переданной вершины.
- fixHeight(node*) - восстанавливает корректное значение поля height заданного узла (при условии, что значения этого поля в правом и левом дочерних узлах являются корректными).

Для балансировки узлов были реализованы методы rotateLeft(node*) и rotateRight(node*). Эти методы могут пригодится в том случае, когда после добавления или удаления вершин возникает расбалансировка поддерева. В этом случае баланс фактор некоторых узлов может оказаться равным 2 или -2. Метод, выполняющий балансировку — balance(node*). Метод проверяет фактор баланс переданной вершины на равность 2, если !=, то просто возвращаем вершину. Если равно == 2, то дальше проверяются фактор балансы сыновей текущей вершины. Если у правого < 0, то выполняется правый поворот, после чего выполняется поворот влево. Также симметрично, в случае, если баланс фактор переданной вершины == -2. Метод print(node*, int) — осуществляет печать дерева в консоль. Все перечисленные методы находятся в поле видимости private, в public созданы обертки над этими методами: insert(int), search(int), showTree(), в которых уже запускаются рабочие методы, в которые

передается корень дерева. Алгоритмы работы insert и find описаны выше в разделе **Теория (2).**

Метод, демонстрирующий уровни дерева — print(node*, int). Принимает указатель на узель и количество печатаемых пробелов в консоль. Функция рекурсивная и работает так: в самом начале ф-ии проверяется, существует ли узел, который в нее передали → после, в переданную переменную spaces прибавляется константное значение добавляемых пробелов → вызывается print для правого потомка →после прохода (после возврата с уровня, где указатель был 0) начинается цикл с диапазоном [spacesCount(const), spaces) в котором печатаются пробелы → дальше идут проверки узлов на булевские переменные inserted или catched (inserted становится true, когда мы вставляем вершину; catched на каждом шаге методом insert и find текущая переданная вершина. Если это «вставленная» вершина, то печатается вершина желтым цветом; если «найденная», то фиолетовым; иначе обычная печать в консоль → вызывается print для левого потомка.

В функции main(int argc, char* argv[]) создается объект AVL_Tree. После чего, проверяется параметр argc больше ли он 1. Если это так, то в цикле заносятся вершины в дерево, на каждом шаге показывая, как происходит вставка. После вставки всех вершин из параметров консоли, программа будет запрашивать команды в режиме runtime. Допустимые команды: insert <vertex_key>, find <vertex_key>. На каждом шаге алгоритмов, выводятся сообщения о происходящем, а также подсвечиваются вершины: текущая вершина на шаге будет подсвечена фиолетовым цветом с подчеркиванием; в случае вставки, вставляемая вершина будет подсвечена желтым цветом.

4. Тестирование и демонстрация.

1) Ввод вершин в консоли.

Для демонстрации были взяты вершины: 15, 10, 20, 22, 5, 2.

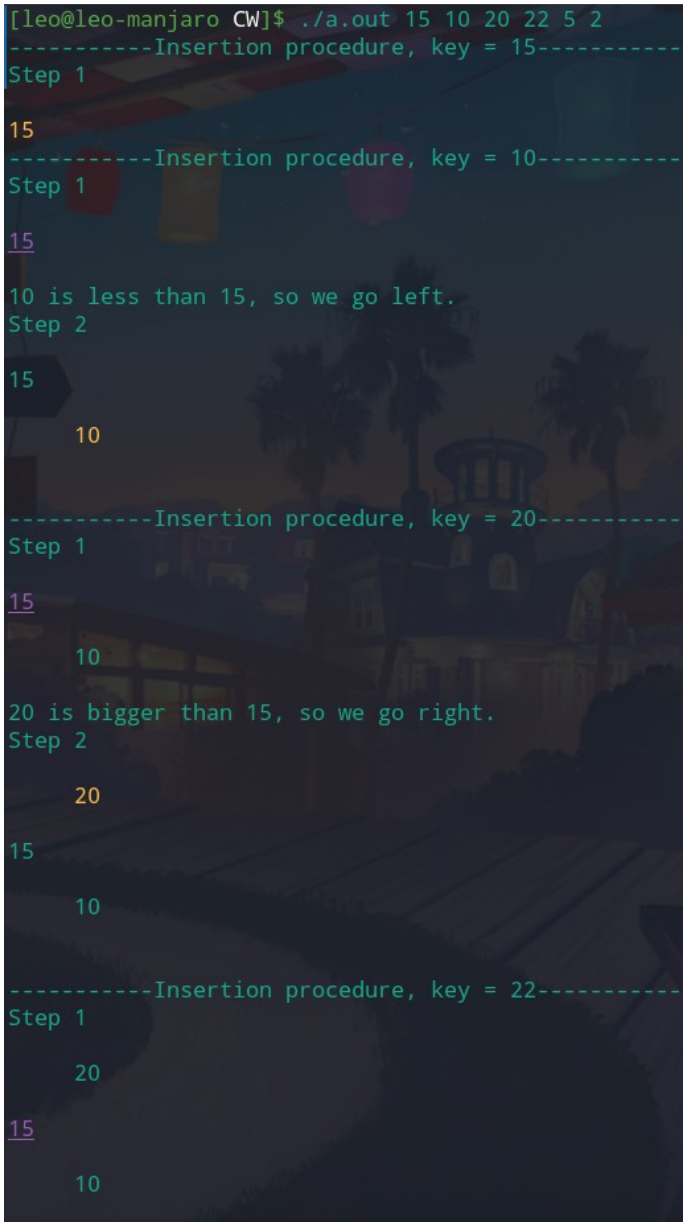


Рисунок 1.1 — Вставка из консоли

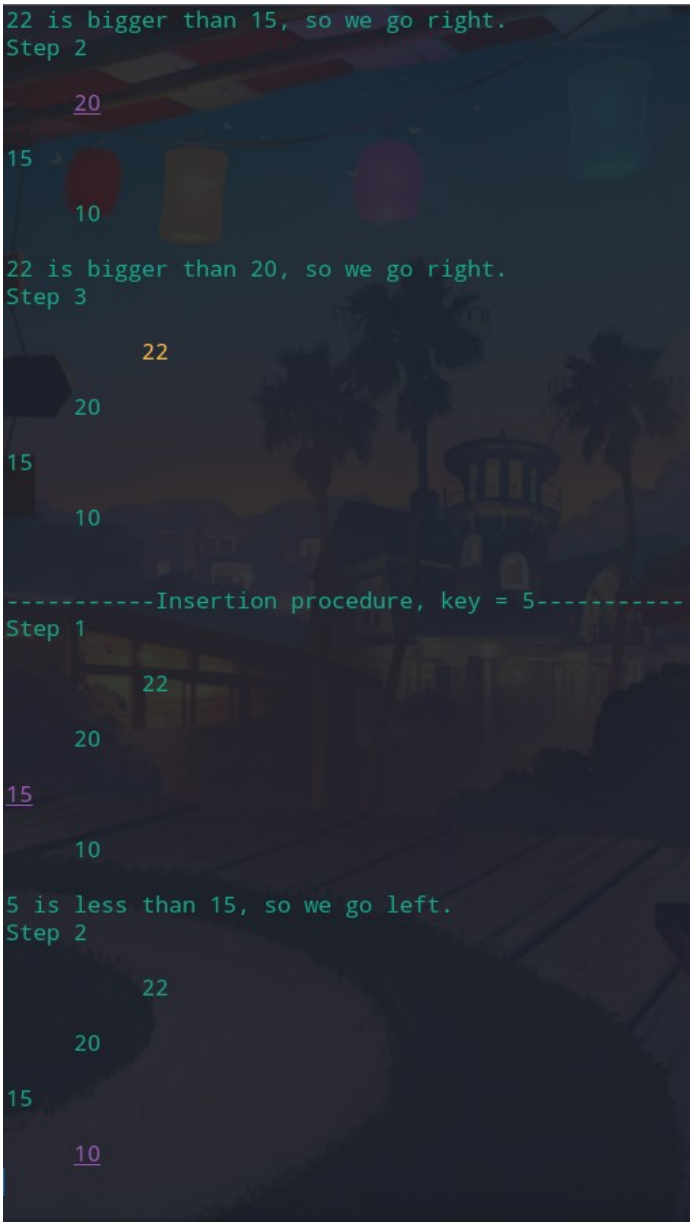


Рисунок 1.2 — Вставка из консоли

2) Вставка вершины через команду.

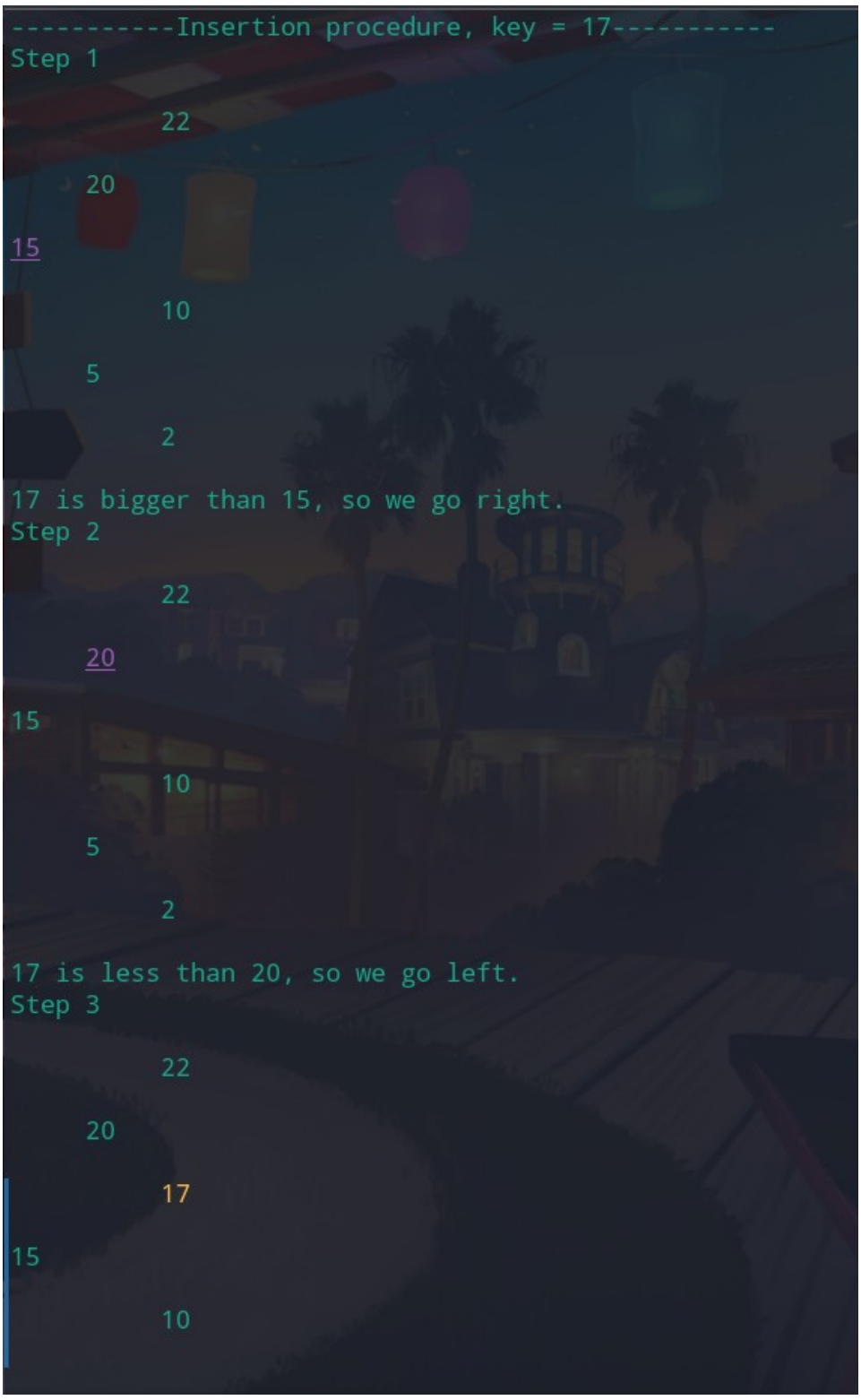


Рисунок 2 — Вставка через команду программы

3) Поиск вершины.

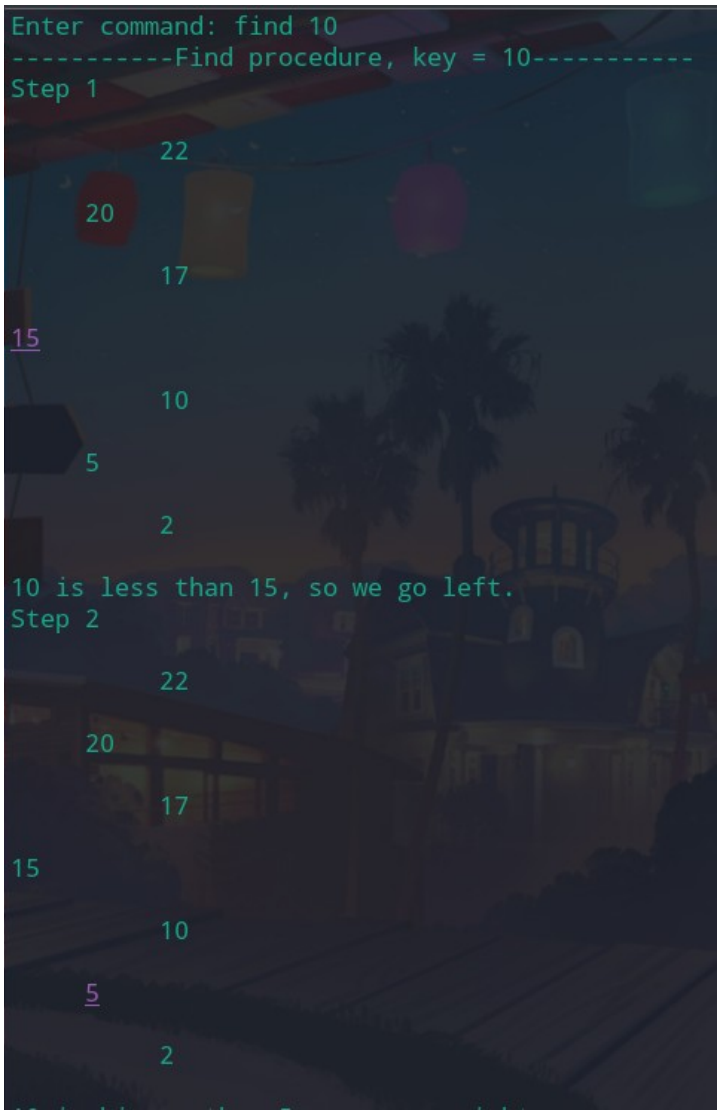


Рисунок 3.1 — Поиск вершины

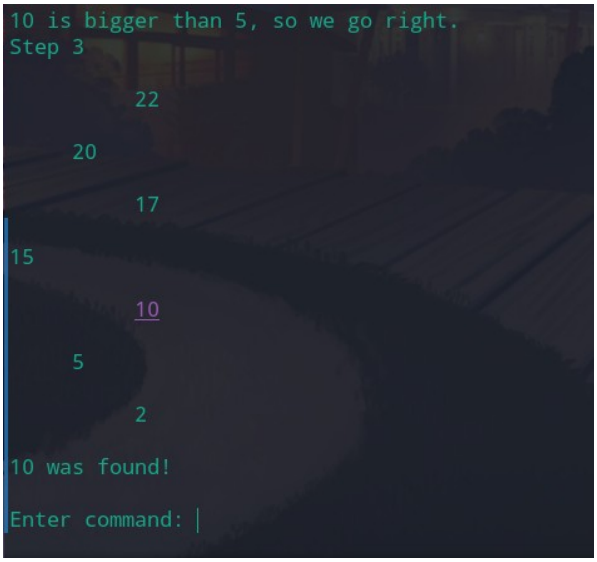


Рисунок 3.2 — Поиск вершины

4. ЗАКЛЮЧЕНИЕ

Была изучена и реализована структура данных АВЛ-дерево со всеми нужными методами для поддержания ее сбалансированной. Также, был реализован один из вариантов, демонстрирующий пошаговый процесс вставки и поиска в СД в runtime режиме.

СД работает корректно и ее можно использовать в будущих проектах.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. https://ru.wikipedia.org/wiki/%D0%90%D0%92%D0%9B-%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE
2. https://habr.com/ru/post/150732/
3. https://neerc.ifmo.ru/wiki/index.php?title=%D0%90%D0%92%D0%9B-%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE
4. https://www.youtube.com/watch?v=gGatm5G1Iy8