МИНОБРНАУКИ РОССИИ
Санкт-Петербургский государственный
электротехнический университет «ЛЭТИ»
им. В.И. Ульянова (Ленина)
Кафедра САПР

ОТЧЁТ
ЛАБОРАТОРНАЯ РАБОТА №3

по дисциплине «Алгоритмы и структуры данных»

Тема: Двоичные деревья (вариант 1)

Студент гр. 9892 _____ Лескин К.А.

Преподаватель _____ Тутуева А.В.

Санкт-Петербург
2021

Задача

Требуется реализовать класс двоичного дерева поиска.
Класс должен включать следующие методы:

1. `bool contains(int)` — поиск элемента в дереве по ключу
2. `void insert(int)` — добавление элемента в дерево по ключу. Должен работать за O(logN)
3. `void remove(int)` — удаление элемента дерева по ключу
4. `Iterator create_dft_iterator()` — создание итератора, реализующего один из методов обхода в глубину (depth-first traverse)
5. `Iterator create_bft_iterator()` — создание итератора, реализующего методы обхода в ширину (breadth-first traverse)

Оценить временную сложность реализуемых метеодов.
Написать unit-тесты.

Описание реализуемого класса и методов

Для выполнения задания будут реализованы два класса.

`BstNode` будет являться узлом дерева. Каждый узел дерева является объектом класса `BstNode`. `BstNode` содержит в себе данные и ссылки на левого и правого потомков.

Основной класс `BST` предназначен для работы с деревом. В нём будут реализованы все методы, требуемые в задании. Класс `BST` содержит только ссылку на корневой узел дерева.

Для выполнения задания используется язык `Python3` версии `3.9`

Дополнительно будут реализованы следующие методы:

`def get(self, value, default=None) -> Optional[BstNode]:` — Метод для получения узла дерева по значению.

`def get_parent(self, value, default=None) -> Optional[BstNode]:` — Метод для получения родительского узла к узлу с переданным значением.

`def _height(self, node) -> int:` — Метод для получения высоты дерева.

`def get_str(self, node: BstNode = _MISSING, indent=0, arrow='--->') -> Generator:` — Метод для получения строки, отображающей дерево.

`def __check_type(value: Any, t: Type = int) -> None:` — Метод для проверки `value` на соответствие типу `t`.

`def get_max_node(node) -> Optional[BstNode]:` — Метод для получения узла с максимальным значением в дереве с корнем в `node`.

Оценка сложности

Метод	Оценка сложности
`contains(self, value: int) -> bool:`	$O(\log n)$
`insert(self, value: int) -> BstNode:`	$O(\log n)$
`remove(self, value: int) -> None:`	$O(\log n)$
`get(self, value, default=None) -> Optional[BstNode]:`	$O(\log n)$
`get_parent(self, value, default=None) -> Optional[BstNode]:`	$O(\log n)$
`dft(self, type: str = 'inorder') -> Generator:`	-
`bft(self) -> Generator:`	$O(n)$
`_inorder(self, node) -> Generator:`	$O(n)$
`_preorder(self, node) -> Generator:`	$O(n)$
`_postorder(self, node) -> Generator:`	$O(n)$
`_height(self, node) -> int:`	$O(n)$
`get_max_node(node) -> Optional[BstNode]:`	$O(\log n)$

Тесты

Были реализованы следующие тесты:

`bst` — тест, проверяющий корректное поведение при выполнении метода .

`test_bst_contains` — тест, проверяющий корректное поведение при выполнении метода `contains`.

`test_bst_contains_with_exception` — тест, проверяющий, что произошло ожидаемое исключение при выполнении метода `contains`.

`test_bst_insert` — тест, проверяющий корректное поведение при выполнении метода `insert`.

`test_bst_insert_with_exception` — тест, проверяющий, что произошло ожидаемое исключение при выполнении метода `insert`.

`test_bst_remove` — тест, проверяющий корректное поведение при выполнении метода `remove`.

`test_bst_remove_with_exception` — тест, проверяющий, что произошло ожидаемое исключение при выполнении метода `remove`.

`test_bst_bft` — тест, проверяющий корректное поведение при выполнении метода `bft`.

`test_bst_dft` — тест, проверяющий корректное поведение при выполнении метода `dft`.

Пример работы

На рисунке 1 прдставлен пример выпонения программы

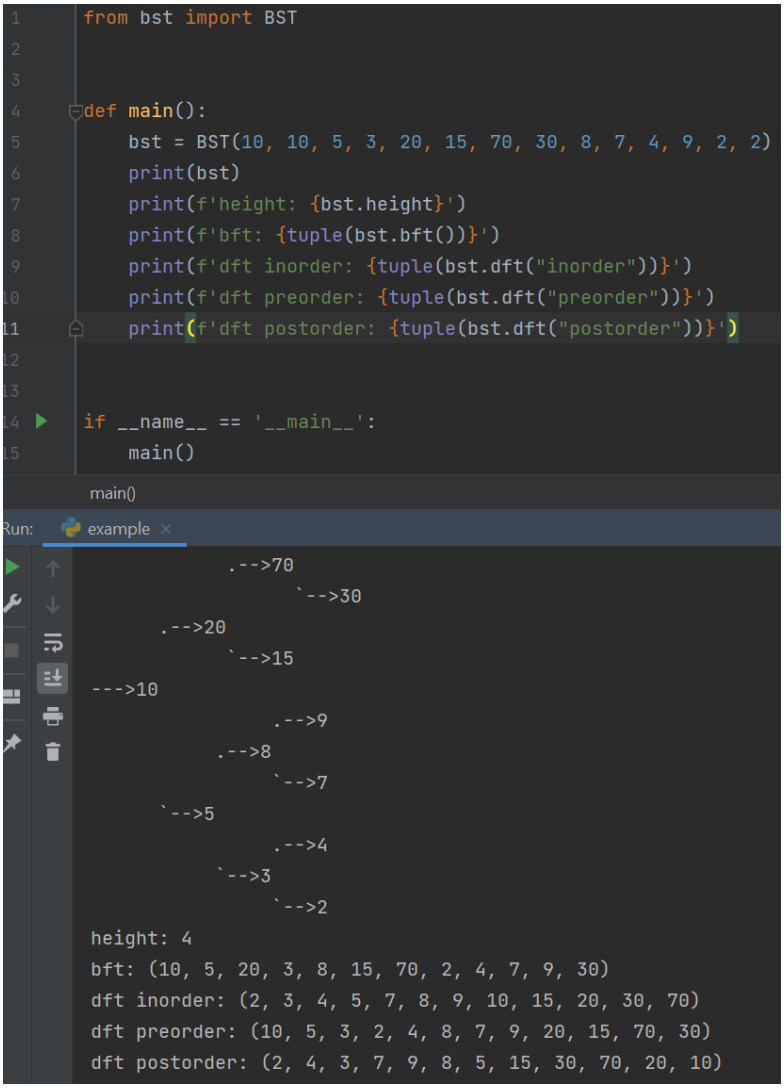

Рис. 1 – Пример выполнения демонстрационной программы

Листинг

Исходный код программы доступен по ссылке:
https://github.com/kira607/3lab-algo-3-1