

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ**
федеральное государственное автономное образовательное учреждение высшего
образования
«Национальный исследовательский технологический университет «МИСИС»
**СТАРООСКОЛЬСКИЙ ТЕХНОЛОГИЧЕСКИЙ ИНСТИТУТ
ИМ. А.А. УГАРОВА**
(филиал) федерального государственного автономного образовательного учреждения
высшего образования
«Национальный исследовательский технологический университет «МИСИС»
(СТИ НИТУ «МИСИС»)
**ФАКУЛЬТЕТ АВТОМАТИЗАЦИИ И ИНФОРМАЦИОННЫХ
ТЕХНОЛОГИЙ КАФЕДРА АВТОМАТИЗИРОВАННЫХ И
ИНФОРМАЦИОННЫХ СИСТЕМ УПРАВЛЕНИЯ
ИМ. Ю.И. ЕРЕМЕНКО**

Домашняя работа №2
по дисциплине: «Теория алгоритмов и структур данных»

Выполнил студент группы: АТ/МС-23Д, Небольсин Василий Дмитриевич

группа, ФИО полностью

подпись

Проверил: ассистент кафедры АИСУ, Жуков Петр Игоревич

Должность, звание, ФИО полностью

подпись

Старый Оскол, 2023

Задание

Реализовать Binary Search Tree со значениями от 1 до 20 и алгоритм поиска в глубину.

Решение

Реализация дерева поиска идентична связному списку с единственным отличием: узел списка указывает на один другой узел, а дерево на большее их число. В последних каждый родительский узел может иметь несколько узлов-потомков. Если у каждого узла максимум два узла-потомка (левый и правый), такое дерево называется двоичным или бинарным.

Класс узла будет содержать в себе три атрибута: значение узла, правый и левый указатель.

Listing 1: Реализация узла дерева поиска

```
class Node:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None
```

Бинарное дерево поиска следует определенному порядку расположения элементов. В дереве двоичного поиска значение левого узла должно быть меньше родительского узла, а значение правого узла больше. Это правило применяется рекурсивно к левому и правому поддеревьям корня.

Listing 2: Вставка элемента в бинарное дерево

```
def insert(root, value):
    if root is None:
        return Node(value)
    elif value < root.value:
        root.left = insert(root.left, value)
    else:
        root.right = insert(root.right, value)
    return root
```

Рассмотрим алгоритм создание бинарного дерева поиска.

Listing 3: Алгоритм создания BST

```
def create_binary_tree(values):
    root = Node(values[0])
    for value in values[1:]:
        insert(root, Node(value))
    return root
```

Сначала нужно вставить корень дерева. Затем прочитать следующий элемент. Если он меньше корневого узла, вставить его как корень левого поддерева и перейти к следующему элементу, в противном случае как корень правого поддерева.

На этом создание бинарного дерева поиска завершено. Теперь перейдем к алгоритмам поиска, которые можно выполнять с BST.

Самые простые в реализации обхода дерева — прямой (Pre-Order), обратный (Post-Order) и центрированный (In-Order), также поиск в ширину и поиск в глубину.

При обходе в глубину (Depth First Search, DFS) алгоритм сначала опускается к низу дерева, а потом идет в сторону, а при обходе в ширину (Breadth First Search, BFS) — наоборот, начинает с корня и обходит узлы-потомки, потом спускается к потомкам потомков, и так далее.

Listing 4: Алгоритм поиска в глубину

```
def depth_first_search(node, value):
    if node is None or node.value == value:
        return node
    elif value < node.value:
        return depth_first_search(node.left, value)
    else:
        return depth_first_search(node.right, value)
```

Результаты

В отличие от массивов и связанных списков двоичное дерево поиска имеет ряд преимуществ.

Операции с деревом работают быстрее. При реализации списком все функции требуют $O(n)$ действий, где n — размер структуры. Операции с деревом же работают за $O(h)$, где h — максимальная глубина дерева (глубина — расстояние от корня до вершины).

В оптимальном случае, когда глубина всех листьев одинакова, в дереве будет $n = 2^h$ вершин. Значит, сложность операций в деревьях, близких к оптимуму будет $O(\log(n))$.

К сожалению, в худшем случае дерево может выродиться и сложность операций будет как у списка, например если вставлять числа $1..n$ по порядку.

Однако существуют способы реализовать дерево так, чтобы оптимальная глубина дерева сохранялась при любой последовательности операций. Такие деревья называют сбалансированными. К ним например относятся красно-черные деревья, AVL-деревья, splay-деревья