BST – PODSTAWOWE OPERACJE | JAVA

DRZEWA BST

Drzewa BST (*Binary Search Tree*) – specjalny rodzaj drzew, pozwalający na optymalne wykonywanie operacji (takich jak wstawianie, szukanie lub usuwanie elementu) na tej strukturze. Są to drzewa binarne (każdy rodzic posiada nie więcej niż dwójkę dzieci) z dodatkową własnością obowiązującą w każdym węźle tej struktury – wartość w węźle nie może być mniejsza od wartości lewego dziecka tego węzła oraz nie jest większa od wartości prawego dziecka tego wierzchołka.

# IMPLEMENTACJA BST W JAVIE

Klasa BST reprezentująca *Binary Search Tree* powinna zawierać następujące składowe:

* wewnętrzną klasę Node, która z kolei zawiera:
  + int key – klucz (wartość) przechowywana w węźle
  + Node left – „wskaźnik” do lewego dziecka
  + Node right – wskaźnik do prawego dziecka
  + Node(int k) – konstruktor, który tworzy węzeł o danej wartości, a left i right ustawia na null (nowo powstały węzeł nie ma jeszcze dzieci)

    class Node {

        int key;

        Node left, right;

        public Node(int k) {

            key = k;

            left = right = null;

        }

    }

* Node root – drzewo BST potrzebuje mieć dostęp do swojego korzenia

    Node root;

* BST() – konstruktor ustawiający root na null

    BST() {

        root = null;

}

* metody wykonujące odpowiednie operację na drzewie BST. Podstawowe z nich to:
  + inorder(Node node) – wypisuje elementy w porządku inorder (dla każdego węzła najpierw wypisuje wszystkie wartości z jego lewego poddrzewa, później klucz węzła, a następnie wszystkie wartości z jego prawego poddrzewa)
  + preorder(Node node) - wypisuje elementy w porządku preorder (dla każdego węzła najpierw wypisuje klucz węzła, później wszystkie wartości z jego lewego poddrzewa, a następnie wszystkie wartości z jego prawego poddrzewa)
  + postorder(Node node) - wypisuje elementy w porządku postorder (dla każdego węzła najpierw wypisuje wszystkie wartości z jego lewego poddrzewa, później wszystkie wartości z jego prawego poddrzewa, a następnie klucz węzła)
  + insert(int key) – wstawia element o kluczu key do drzewa
  + delete(int key) – usuwa element o kluczu key z drzewa (o ile istnieje)
  + search(int key) – szuka elementu o kluczu key (jeżeli znajdzie zwróci ten węzeł, a jeśli nie zwróci null)

METODA INORDER()

    void inorder(Node root) {

        if(root != null) {

            inorder(root.left);

            System.out.printf(root.key + " ");

            inorder(root.right);

        }

    }

METODA PREORDER()

    void preorder(Node root) {

        if(root != null) {

            System.out.printf(root.key + " ");

            preorder(root.left);

            preorder(root.right);

        }

    }

METODA POSTORDER()

    void postorder(Node root) {

        if(root != null) {

            postorder(root.left);

            postorder(root.right);

            System.out.printf(root.key + " ");

        }

    }

METODA INSERT()

    void insert(int k) {

        if(root == null) {

            root = new Node(k);

            return;

        }

        Node current\_node = root;

        while((k < current\_node.key&&current\_node.left != null)||(k >= current\_node.key&&current\_node.right != null)) {

            if(k < current\_node.key) {

                current\_node = current\_node.left;

            } else {

                current\_node = current\_node.right;

            }

        }

        if(k < current\_node.key) {

            current\_node.left = new Node(k);

        } else {

            current\_node.right = new Node(k);

        }

    }

METODA SEARCH()

Node search(Node root, int k)

    {

        if(root == null || root.key == k)

            return root;

        if(root.key < k)

            return search(root.right, k);

        return search(root.left, k);

    }