



# Politechnika Poznańska

Informatyka rok I semestr 2

L10, Piątek 11:45 - 13:15

## Algorytmy i Struktury Danych

Prowadzący: Dominik Piotr Witczak

### Sprawozdanie nr 2

Drzewa przeszukiwań binarnych  
BST i drzewa samobalansujące AVL

**Autor:**

Dominik Fischer 164176

Oliwer Miller 163544

Rok akademicki 2024/2025

# Wprowadzenie

## Tworzenie drzewa BST

Klasa *BSTNode* reprezentuje pojedynczy węzeł drzewa BST. Składa się z klucza *key* i odnośników do lewego i prawego potomka.

Klasa *BST* reprezentuje drzewo BST. Korzeń drzewa jest zapisany w *root*. Funkcja *insert* służy do dodawania nowych węzłów, wykorzystuje do tego *\_insert*, które rekurencyjnie wstawia węzeł po lewej lub prawej stronie, zależnie od jego wartości.

### Terminal

```
class BSTNode:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None

class BST:
    def __init__(self):
        self.root = None

    def insert(self, key):
        self.root = self._insert(self.root,
                                   key)

    def _insert(self, node, key):
        if not node:
            return BSTNode(key)
        if key < node.key:
            node.left = self._insert(node.left,
                                      key)
        elif key > node.key:
            node.right = self._insert(node.
                                       right, key)
        return node
```



## Tworzenie drzewa AVL

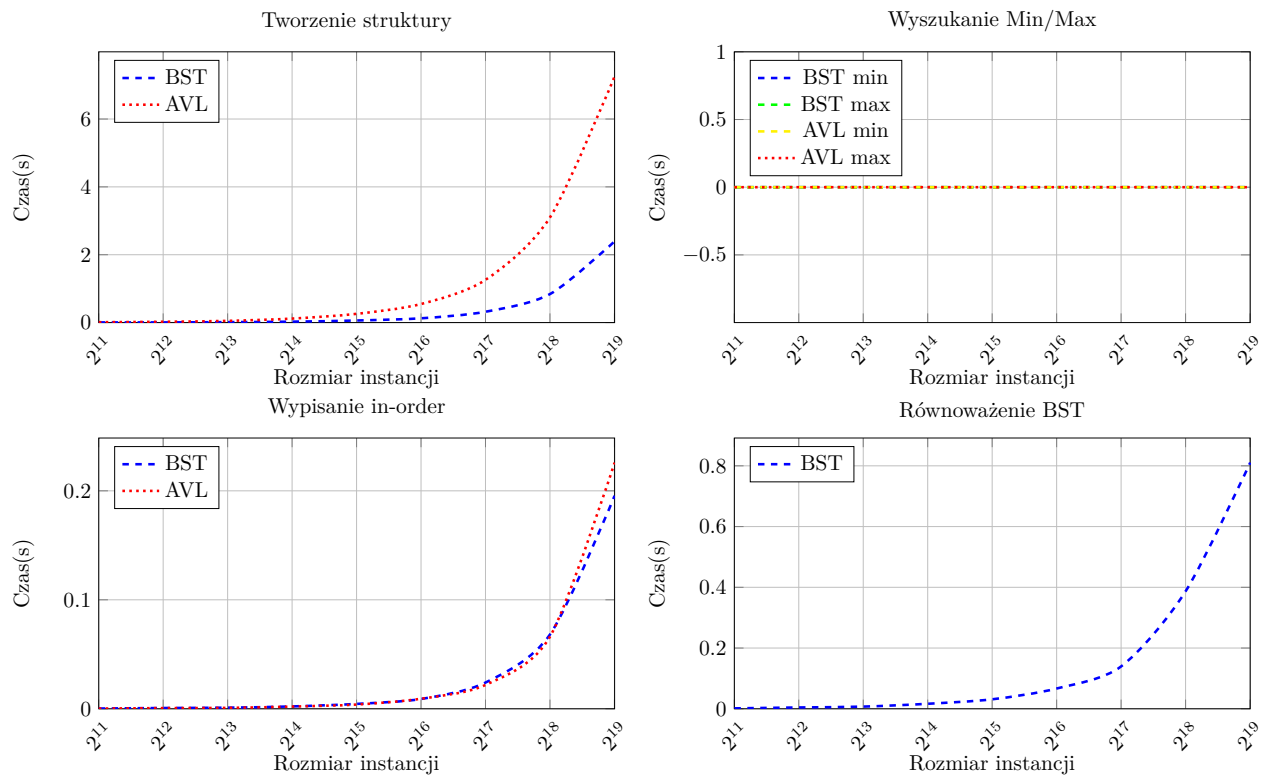
tekst

### Terminal

```
class AVLNode:
    def __init__(self, key):
        self.key = key
        self.height = 1
        self.left = None
        self.right = None

class AVL:
    def __init__(self):
        self.root = None
    def build_from_sorted(self, values):
        def build(vals):
            if not vals:
                return None
            mid = len(vals) // 2
            node = AVLNode(vals[mid])
            node.left = build(vals[:mid])
            node.right = build(vals[mid+1:])
            node.height = 1 + max(self.get_height(
                node.left), self.get_height(node.
                right))
            return node
        self.root = build(values)
    def get_height(self, node):
        return node.height if node else 0
    def get_balance(self, node):
        return self.get_height(node.left) -
            self.get_height(node.right) if
            node else 0
    def update_height(self, node):
        node.height = 1 + max(self.get_height(
            node.left), self.get_height(node.
            right))
    def rotate_right(self, y):
        x = y.left
        T2 = x.right
        x.right = y
        y.left = T2
        self.update_height(y)
        self.update_height(x)
        return x
    def rotate_left(self, x):
        y = x.right
        T2 = y.left
        y.left = x
        x.right = T2
        self.update_height(x)
        self.update_height(y)
        return y
    def insert(self, key):
        self.root = self._insert(self.root,
            key)
    def _insert(self, node, key):
        if not node:
            return AVLNode(key)
        if key < node.key:
            node.left = self._insert(node.left,
                key)
```

## Wykresy



Rysunek 1: Wykresy tworzenia, wyszukiwania min/max, wypisania in-order, równoważenia