Laboratoria Algorytmów i Struktur Danych

Sprawozdanie nr 2

Drzewa przeszukiwań binarnych BST drzewa samobalansujące

Prowadzący: **Dominik Piotr Witczak**

Autorzy: **Marcin Jakubowski(164108), Adam Woźiński(164119)**

POLITECHNIKA POZNAŃSKA

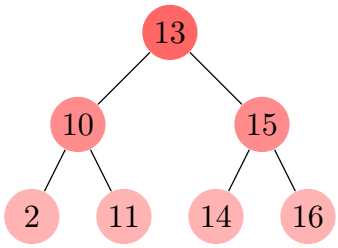
1 | Wprowadzenie

Celem projektu było zaimplementowanie dwóch struktur danych: drzewa binarnego poszukiwań (BST - Binary Search Tree) oraz drzewa AVL (AVL Tree), a następnie porównanie ich działania w różnych sytuacjach. Drzewa te są podstawowymi strukturami danych w informatyce, wykorzystywanymi w wielu dziedzinach, takich jak bazy danych, systemy plików czy kompilatory.
Drzewo BST pozwala na przechowywanie danych w uporządkowany sposób, co umożliwia szybkie wyszukiwanie, wstawianie i usuwanie elementów. Jednak w przypadku niekorzystnych danych wejściowych, np. posortowanych, drzewo może stać się niezrównoważone, co znacząco obniża jego wydajność.
Drzewo AVL to ulepszona wersja drzewa BST, która automatycznie utrzymuje balans. Dzięki temu wysokość drzewa pozostaje logarytmiczna względem liczby elementów, co zapewnia wysoką wydajność operacji niezależnie od danych wejściowych. Balansowanie odbywa się za pomocą rotacji (lewych i prawych), które są wykonywane, gdy różnica wysokości poddrzew przekracza dopuszczalny limit.
W ramach projektu zaimplementowano podstawowe operacje na drzewach, takie jak wstawianie, usuwanie czy wyszukiwanie wartości minimalnej i maksymalnej. Dodatkowo dodano funkcje balansowania drzewa BST oraz możliwość wizualizacji struktury drzewa. Program umożliwia także interakcję z drzewami za pomocą prostego interfejsu tekstowego, co pozwala na testowanie i analizę ich działania.

2 | Struktura drzewa

Terminal

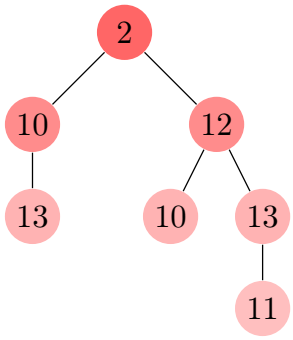
```
# Tworzenie drzewa AVL
class AVLNode:
def __init__(self, key):
self.key = key
self.left = None
self.right = None
self.height = 1
```



(a) Tworzenie drzewa AVL

Terminal

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



(b) Tworzenie drzewa BST

3 | Tworzenie drzew

Proces rozpoczyna się od wprowadzenia danych za pomocą heredoc. Następnie użytkownikowi prezentowane jest główne menu, które umożliwia wybór dostępnych opcji związanych z dalszym zarządzaniem drzewami.

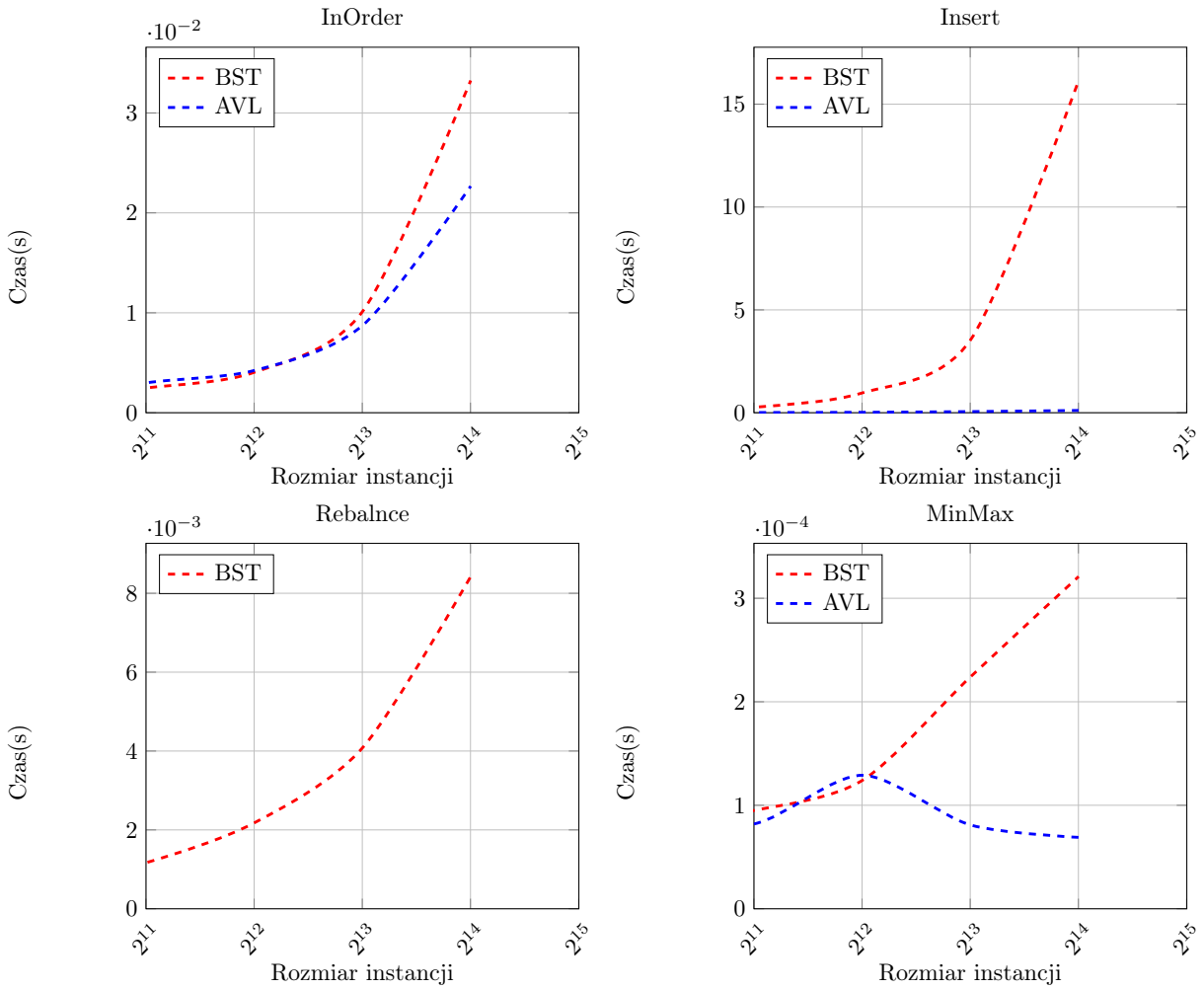
Terminal

```
python3 ./main.py --tree BST < test.txt
Enter values to insert into the BST tree:
insert> 1 2 3 4 5 6 7 8 9 12 13
Tree is completed


    Help          Show this message
    Print         Print the tree usin In-order, Pre-order, Post-order
    FindMinMax    Find the minimum and maximum values in the tree
    Draw          Draw the tree
    Remove        Remove elements of the tree
    Delete All    Delete whole tree
    Rebalance     Rebalance the tree
    Exit          Exit the program (same as Ctrl+D)

action>
```

4 | Porównywanie czasów wykonywania operacji



5 | Wnioski

Drzewa AVL są szybsze od drzew BST w większości operacji, szczególnie przy dużych zbiorach danych. AVL utrzymuje równowagę drzewa, co pozwala na lepszą wydajność, podczas gdy BST może się "wydłużać", co spowalnia działanie.
Wstawianie w AVL wymaga dodatkowego czasu na utrzymanie równowagi, ale przy większych danych AVL działa szybciej niż BST. Automatyczne równoważenie w AVL pozwala na lepszą wydajność w dłuższej perspektywie, czego BST nie oferuje.
Operacje takie jak przejście drzewa w porządku czy znajdowanie wartości minimalnej i maksymalnej są szybsze w AVL dzięki mniejszej głębokości drzewa. W BST te operacje mogą być wolniejsze, jeśli drzewo jest niezrównoważone.
Im większy zbiór danych, tym bardziej AVL przewyższa BST. AVL jest lepszym wyborem do dużych zbiorów danych, gdzie kluczowa jest wydajność, natomiast BST sprawdzi się w prostszych przypadkach.