Sprawozdanie
AISDI, **drzewa**

wykonujący:
Jakub Rozkosz
Hubert Gołębiowski

Podział pracy:
Jakub Rozkosz:

- implementacja drzewa BST
- napisanie funkcji do rysowania wykresów (i implementacja w mainie)
- pisanie sprawozdania

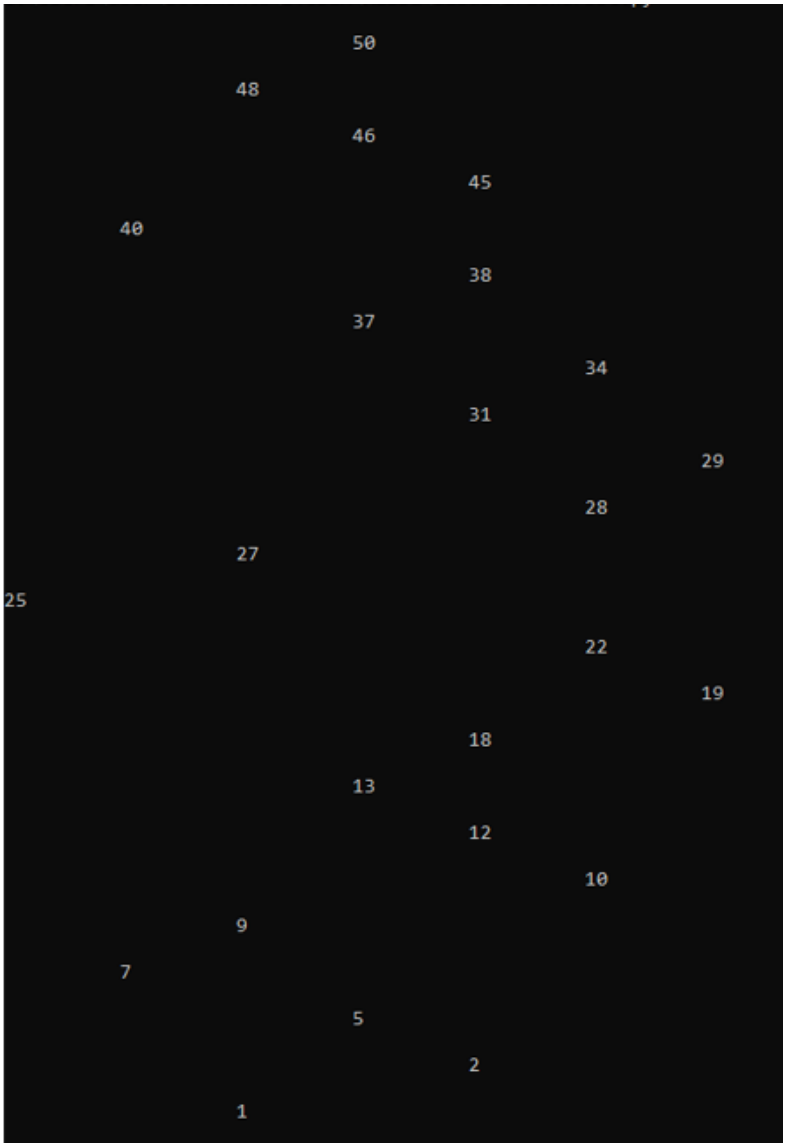
Hubert Gołębiowski:

- implementacja drzewa AVL
- wykonanie pomiarów (i implementacja w mainie)
- pisanie sprawozdania

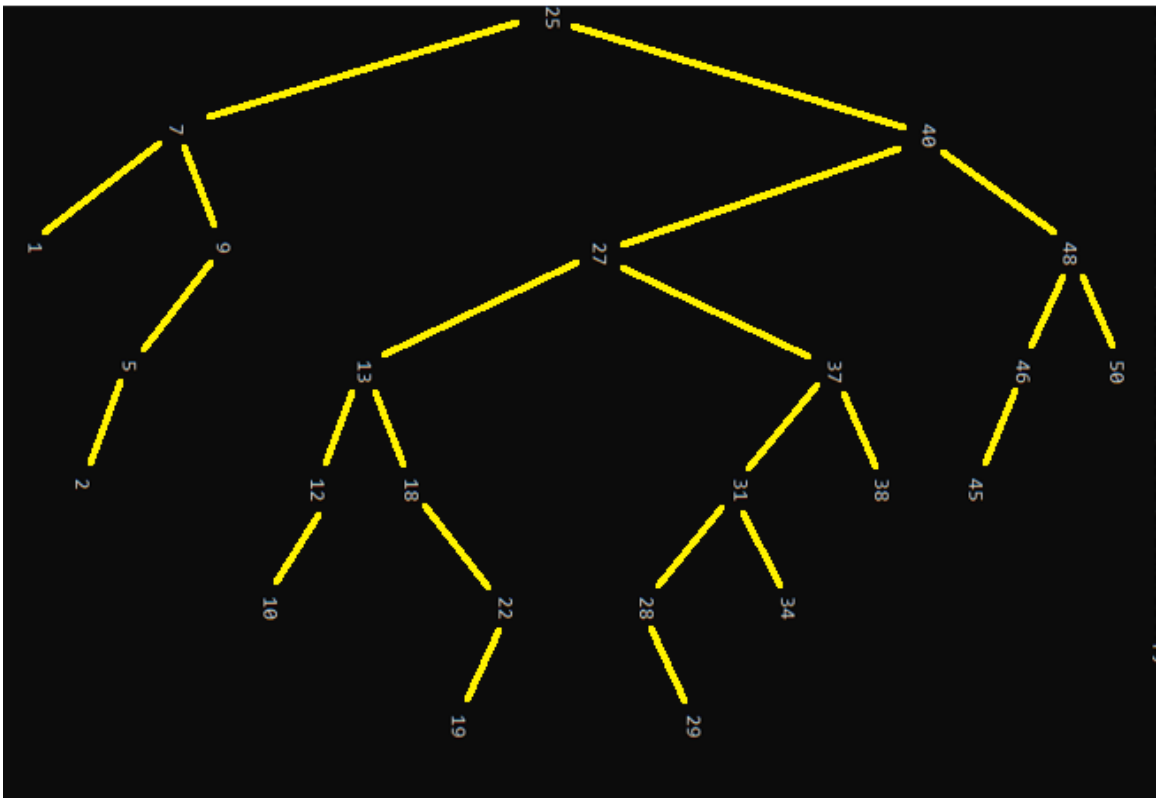
Pliki:
BST.py - plik z implementacją drzewa BST. Zawiera klasę Node, której obiekty są węzłami w drzewie. Klasa BST reprezentuje całe drzewo - jej składową, która tworzona jest przy inicjalizacji obiektu, jest korzeń drzewa. Posiada metody wstawiania (*insert*), usuwania (*delete*), szukania elementu drzewa (*search*) oraz wyświetlanie drzewa (*print_tree*).
AVL.py - plik z implementacją drzewa AVL. Zawiera klasę Node, której obiekty są węzłami w drzewie. Klasa AVL reprezentuje całe drzewo - jej składową, która tworzona jest przy inicjalizacji obiektu, jest korzeń drzewa. Posiada metody wstawiania (*insert*), usuwania (*delete*), szukania elementu drzewa (*find*) oraz wyświetlanie drzewa (*print_tree*).
plots.py - plik z funkcją do rysowania zbiorczych wykresów dla każdej z operacji
main.py - główny plik, w którym wykonywane są pomiary oraz generowane są wykresy
time_measure.py - plik, w którym znajduje się funkcja *getTime*, która wykonuje pomiary czasowe dla każdej z operacji

Opis:
W naszym projekcie porównaliśmy ze sobą czasy wykonywania poszczególnych metod dla obu drzew. Do pomiarów wykorzystaliśmy funkcje process_time() z biblioteki time. Pomiary wykonywaliśmy dla losowo wygenerowanych danych (od 5.000 do 50.000). Najpierw wszystkie elementy listy wstawiliśmy do drzewa, następnie wszystkie wyszukiwaliśmy (w losowej kolejności), a na końcu usuwaliśmy wszystkie elementy z drzewa (również w losowej kolejności). Warto dodać, że zarówno drzewo BST jak i AVL otrzymywały dokładnie taką samą listę argumentów.

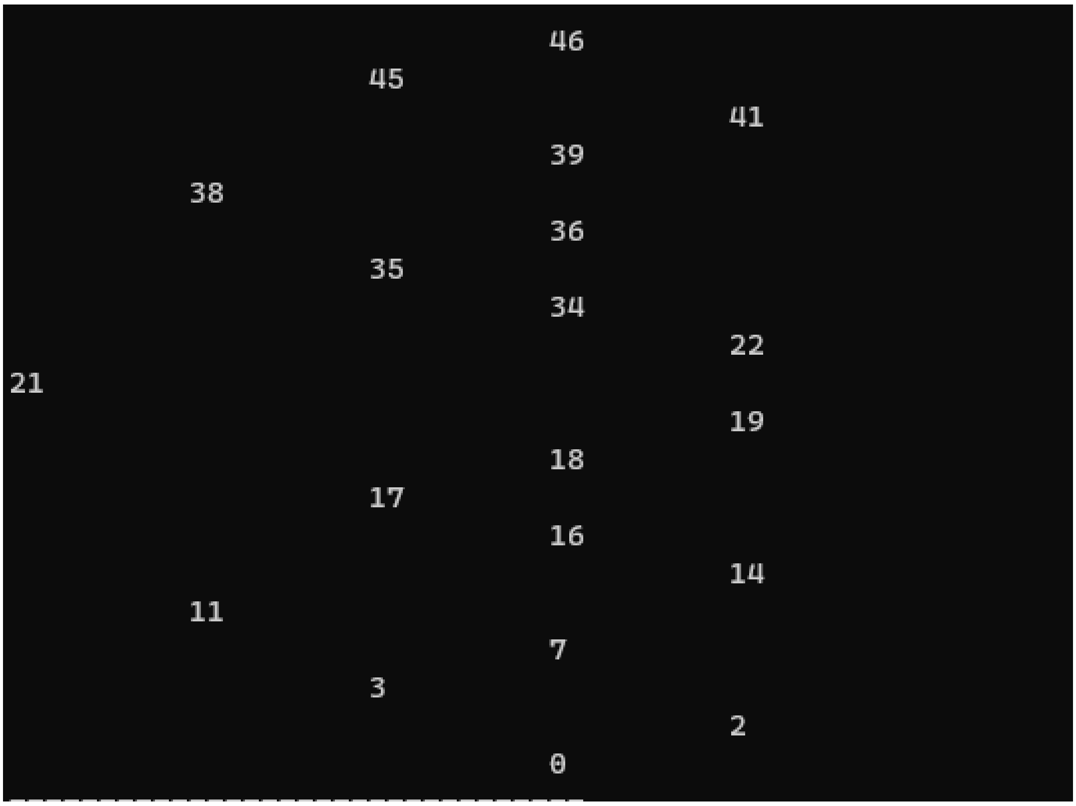
Wyniki:
Drukowanie drzewa BST:
drzewo jest obrócone o 90 stopni w lewo i rozszerza się horyzontalnie



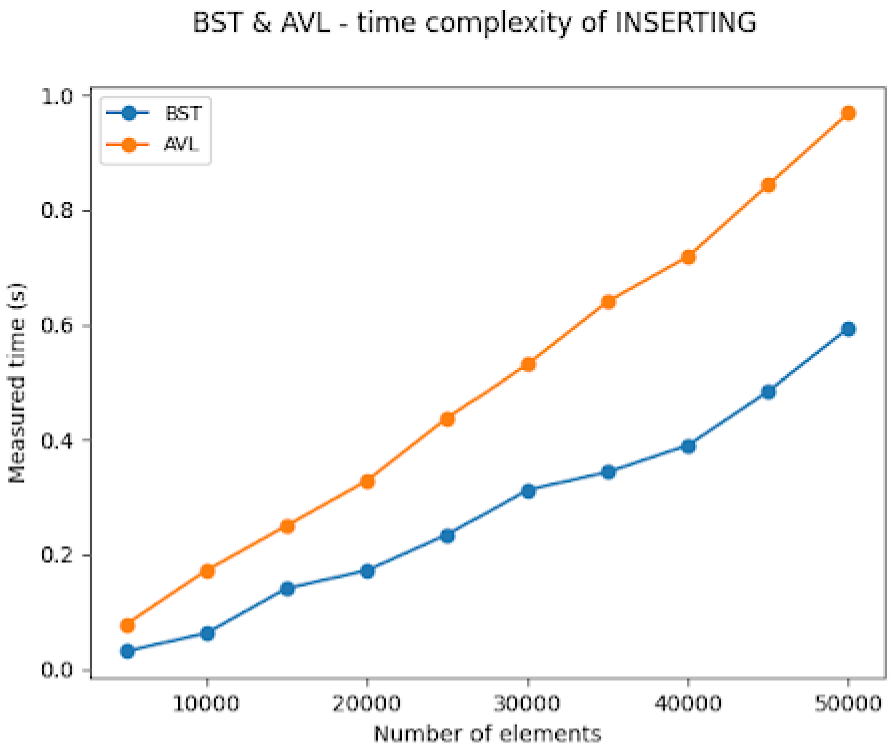
(poniżej jak wygląda drzewo)



Drukowanie drzewa AVL:

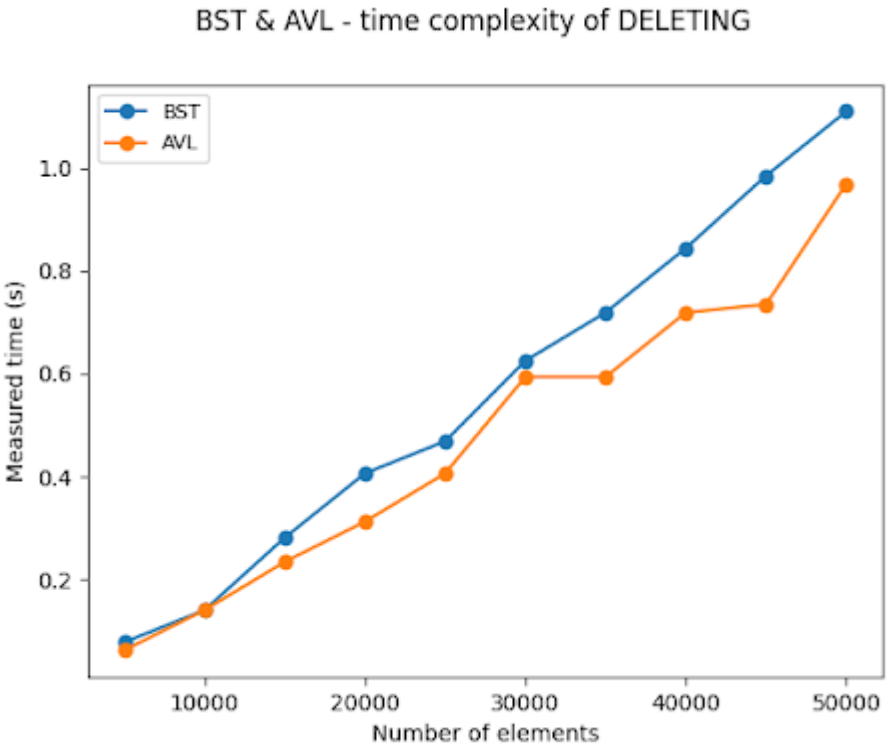


Wstawianie elementów do drzewa:



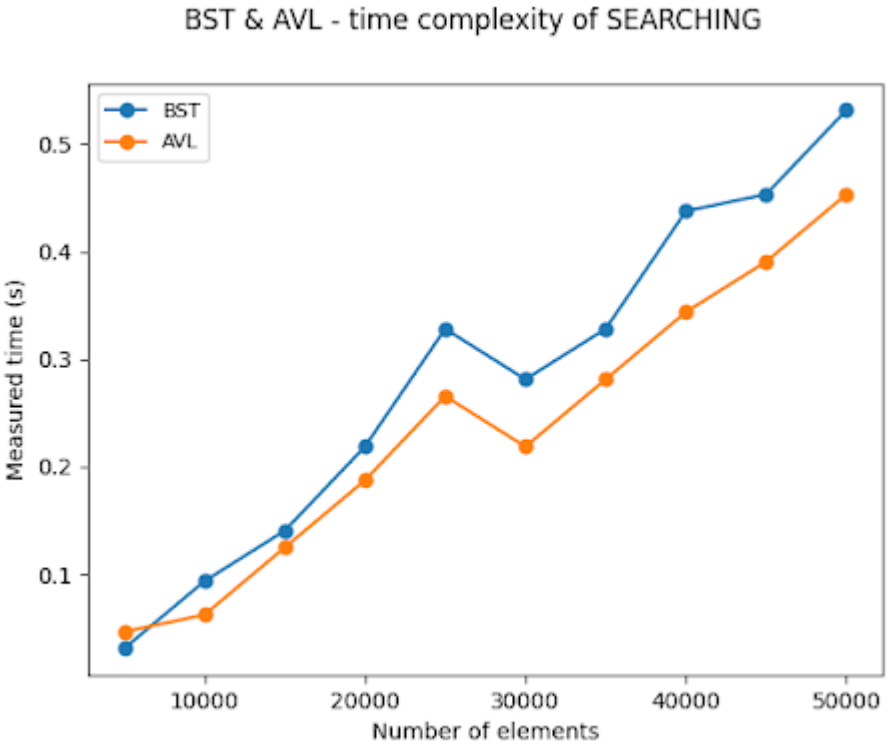
Ewidentnie wstawianie elementów do drzewa BST odbywa się znacznie wolniej. Jest to spowodowane tym, że w drzewie AVL przy każdym wstawianiu musi być ono równoważone, co wiąże się wywołaniami kolejnych funkcji.

Usuwanie elementów z drzewa:



Tutaj różnica jest już dużo mniejsza, natomiast tym razem na minimalną korzyść drzewa AVL. Przy każdym usuwaniu elementu drzewo AVL musi równoważyć poziom (dosyć kosztowne czasowo), natomiast ma ono dużo mniejszą wysokość i dzięki temu nadrabia wobec drzewa BST.

Szukanie elementów w drzewie:



Na wykresie widać, że drzewo AVL jest nieznacznie szybsze od BST w wyszukiwaniu elementów. Spodziewaliśmy się, że AVL będzie miało przewagę, jednak oczekiwaliśmy

trochę większej rozbieżności. Drzewo BST przez to, że nie jest równoważone ma większą wysokość. To wiąże się z dłuższym czasem dotarcia do nisko położonych elementów, które mają zostać usunięte.

Wnioski:

Drzewo AVL ma bardziej kosztowne wstawianie niż drzewo BST, ale za to pozostałe operacje na drzewie są stosunkowo szybsze. Jest to całkiem korzystne rozwiązanie, gdyż zazwyczaj elementy do drzewa wstawia się tylko raz, a wyszukuje niezliczoną ilość razy, dzięki czemu możemy zaoszczędzić czas. Dodatkowo dostarcza ono czytelniejszą wizualizację drzewa.