# Dokumentacja Projektu: BST DSW

**Autor:** Julia Backa

**Temat:** Implementacja drzewa BST z łączem do rodzica i z algorytmem równoważenia Day–Stout–Warren

**Język:** Python 3

## Spis treści

# 1.Wstęp Teoretyczny

## 1.1. Drzewo BST (Binary Search Tree)

Binarne drzewo poszukiwań (ang. Binary Search Tree, BST) – dynamiczna struktura danych będąca drzewem binarnym, w której dla każdego węzła zachodzi zależność: wartości w lewym poddrzewie są mniejsze, a w prawym większe od wartości węzła. Głównym problemem BST jest tendencja do **degeneracji** (staje się listą), co zwiększa złożoność operacji z O(log n) do O(n).

## 1.2. Algorytm Day–Stout–Warren (DSW)

Algorytm DSW służy do równoważenia drzewa BST. Jego zaletą jest niewielkie i stałe zużycie dodatkowej pamięci oraz brak konieczności sortowania danych ani całkowitej dekompozycji drzewa i jego ponownej rekonstrukcji. Składa się z dwóch głównych faz:

- **Faza 1: Tworzenie kręgosłupa (Vine):** Za pomocą rotacji w prawo przekształcamy drzewo w listę, gdzie każdy węzeł ma tylko prawe dziecko.
- **Faza 2: Kompresja (Compression):** Poprzez serie rotacji w lewo, kręgosłup jest "składany" w idealnie zbalansowane drzewo binarne o minimalnej wysokości.

# 2. Opis Interfejsu

**Program oferuje dwa niezależne interfejsy:**

## 2.1. Interfejs Graficzny (Pygame)

**Jest to główny moduł wizualizacyjny (main_pygame.py).**

- **Klawisz I (Insert):** Aktywuje tryb wprowadzania danych. Po wpisaniu liczby i naciśnięciu Enter, nowy węzeł jest dodawany do drzewa.
- **Klawisz V (Vine):** Uruchamia animowaną fazę tworzenia kręgosłupa.
- **Klawisz B (Balance):** Uruchamia animowaną fazę kompresji (równoważenia).
- **Klawisz C (Clear):** Czyści całe drzewo.
- **Klawisz R (Reset camera):** przywraca domyślną pozycję kamery

- **Mysz:** Lewy przycisk służy do przesuwania kamery (Pan), a kółko (Scroll) do przybliżania i oddalania widoku (Zoom).

### 2.2. Interfejs Konsolowy (CLI)

Moduł bst_dsw.py pozwala na pracę w trybie tekstowym. Wyświetla drzewo w formie grafu ASCII bezpośrednio w terminalu.

**Komendy:**

- **i <val> -** wprowadzanie nowego węzła (np. i 10)
- **v –** faza vine
- **b –** faza kompresji (równoważenie)
- **c –** wyczyść całe drzewo
- **q -** wyłącz

## 3.Uwagi na temat implementacji

Projekt implementuje drzewo BST z łączem do rodzica i realizuje algorytm (Day–Stout–Warren). Całość rozłożona jest na następujące moduły:

- **bst_dsw.py**: Rdzeń projektu – zawiera klasy Node i BST wraz z logiką rotacji i balansu.
- **bst_dsw_text.py**: Interfejs tekstowy pozwalający na testowanie drzewa w konsoli.
- **main_pygame.py**: Graficzna prezentacja działania algorytmu z wykorzystaniem biblioteki Pygame.
- **tests.py**: Zestaw testów jednostkowych sprawdzających poprawność rotacji i balansowania.

W tej sekcji opisano klasy, metody oraz rozwiązania techniczne zastosowane w kodzie.

### 3.1. Klasy i funkcje

- **bst_dsw.py**:

- **Klasa Node**: Reprezentuje węzeł drzewa. Posiada wartość – data, wskaźniki left, right oraz wskaźnik do rodzica – parent. Oprócz tego

posiada atrybuty x, y, które służą do przechowywania współrzędnych obliczonych przez algorytmy rysujące dla main_pygame.py

- **Klasa BST:** Zarządza drzewem jako całością. Kluczowe metody to:
    - **insert(self, data):** Klasyczne wstawianie z zachowaniem porządku BST i przypisaniem parent
    - **rotate_left(self, node) / rotate_right(self, node):** Mechanizm rotacji w prawo i w lewo potrzebny do DSW
    - **make_vine(self, callback):** Przekształcenie dowolnego drzewa w „kręgosłup”
    - **_compress(self, count, callback):** To pomocnicza metoda wykonująca zadaną liczbę (count) rotacji w lewo wzdłuż kręgosłupa
    - **balance_dsw(self, callback)**: Najpierw liczy węzły, a potem oblicza matematycznie, ile rotacji potrzeba, by uzyskać idealną równowagę

- **main_pygame.py:**

  **- update_positions(node, x, y, dx):** Rekurencyjnie wyznacza współrzędne x i y dla każdego węzła

  **- draw_tree(screen, node, off_x, off_y, zoom):** Rekurencyjna funkcja renderująca - najpierw rysuje krawędzie (linie) łączące rodziców z dziećmi, a następnie nakłada na nie węzły, tworząc graf

  **- refresh_screen(tree, msg, input_txt, cam_x, cam_y, zoom):** Metoda zarządzająca czyszczeniem i odświeżaniem bufora ekranu. Łączy w sobie wywołanie aktualizacji pozycji, rysowanie drzewa oraz renderowanie statycznego interfejsu

  **- def animate_step(custom_msg):** Wywołuje refresh_screen, przekazując aktualny komunikat o operacji i wstrzymuje wykonanie programu na krótki czas

## 3.2. Rozszerzona struktura węzła (Parent Links)

Standardowa implementacja drzewa BST opiera się na dwóch wskaźnikach (left, right). W niniejszym projekcie klasa Node została rozszerzona o atrybut parent.

- **Wyzwanie techniczne:** Wprowadzenie wskaźnika do rodzica znacząco komplikuje operacje rotacji. Każda rotacja musi poprawnie zaktualizować nie tylko relacje "rodzic-dziecko", ale również zwrotne powiązanie "dziecko-rodzic".

```
node.left = left_child.right
if left_child.right:
    left_child.right.parent = node
```

- **Zaleta:** Dzięki temu struktura jest bardziej elastyczna i pozwala na szybką nawigację w górę drzewa.

## 3.3. Wykorzystanie węzła Pseudo-korzeń (Pseudo-root)

W metodach make_vine oraz _compress zastosowano technikę węzła pomocniczego, pełniącego rolę tzw. pseudo-korzenia (pseudo_root). Jest to kluczowy element zapewniający stabilność struktury podczas intensywnych rotacji.

- **Cel:** Węzeł ten służy jako stabilny "punkt zaczepienia" umieszczony bezpośrednio nad faktycznym korzeniem drzewa. Dzięki temu, podczas wykonywania rotacji na samym szczycie drzewa, algorytm zawsze posiada węzeł nadrzędny, do którego może "przypiąć" nowo powstały korzeń poddrzewa. W ten sposób rzeczywisty korzeń jest traktowany przez logikę rotacji jak zwykłe prawe dziecko.

```
pseudo_root = Node(None)
pseudo_root.right = self.root
if self.root: self.root.parent = pseudo_root

tail = pseudo_root
```

## 3.4. Logika obliczeń w algorytmie DSW

W metodzie balance_dsw zastosowano podane podejście do obliczeń matematycznych:

- **Wykorzystanie bit_length():** Zamiast kosztownych operacji logarytmicznych, wykorzystano metodę n.bit_length(), aby wyznaczyć liczbę węzłów, które powinny stworzyć najbliższe pełne drzewo binarne (m = 2^k - 1).

```
if n > 0:
    m = 2 ** (n.bit_length() - 1) - 1
    self._compress(n - m, callback)
```

- **Separacja faz:** Algorytm jest wyraźnie podzielony na fazę vine oraz fazę compress, co ułatwia debugowanie i pozwala na wizualizację każdego etapu osobno.

## 3.5. Wsparcie dla wizualizacji (Callback System)

- Metody make_vine, _compress oraz balance_dsw przyjmują opcjonalny parametr callback, który – jeśli zostanie przekazany – jest wywoływany po każdej istotnej operacji strukturalnej, takiej jak rotacja w prawo lub w lewo.

```
        if callback: callback("Balancing: Rotating Left...")
self.root = pseudo_root.right
if self.root: self.root.parent = None
```

W module graficznym main_pygame.py rolę funkcji callback pełni metoda animate_step.

```
def animate_step(custom_msg):
    """
    Odswieza ekran i wstrzymuje dzialanie programu na krotki czas,
    aby wizualizacja zmian w strukturze drzewa byla plynna i widoczna.
    """
    refresh_screen(tree, custom_msg, input_txt: "", cam_x, cam_y, zoom)
    time.sleep(0.3)
```

# 4. Podsumowanie i Wyniki Testów

## 4.1. Metodologia testów

Projekt został poddany trzystopniowej weryfikacji:

1. **Testy jednostkowe (Unit Tests):** Automatyczne sprawdzenie podstawowych funkcji struktury BST.
2. **Testy wydajnościowe (Stress Tests):** Weryfikacja zachowania algorytmu przy dużej skali danych.
3. **Weryfikacja wizualna:** Obserwacja procesu rotacji w module Pygame w celu potwierdzenia zachowania własności BST w każdym kroku animacji.

## 4.2. Wyniki testów jednostkowych

Wykorzystując bibliotekę unittest, potwierdzono poprawność następujących operacji:

- **Wstawianie (Insertion):** Węzły są poprawnie rozmieszczane w drzewie (mniejsze na lewo, większe na prawo).
- **Wskaźniki rodzica (Parent Links):** Po każdej rotacji każdy węzeł posiada poprawny wskaźnik do swojego rodzica, co potwierdza stabilność implementacji metod rotate_left i rotate_right.
- **Balansowanie:** Dla zestawu 7 elementów, algorytm każdorazowo generuje zbalansowane drzewo o wysokości 3 z korzeniem będącym medianą zbioru.

## 4.3. Stress Test (1000 węzłów)

Kluczowym elementem weryfikacji był test obciążeniowy na próbie 1000 losowych liczb całkowitych.

- **Przed balansowaniem:** Drzewo wygenerowane z losowych danych posiadało wysokość znacznie przekraczającą logarytmiczne optimum.
- **Po balansowaniu:** Algorytm DSW zredukował wysokość drzewa do wartości 10. Jest to wynik zgodny z wzorem $\log_2(1000+1) = 10$.

### 4.4. Wnioski końcowe

Projekt zakończył się sukcesem i pozwolił na wyciągnięcie następujących wniosków:

1. **Skuteczność DSW:** Algorytm Day–Stout–Warren jest niezwykle efektywnym narzędziem do równoważenia drzew BST, eliminującym problem ich degeneracji bez użycia dodatkowej pamięci.

# 5. Źródła

## Źródła:


https://achilles.tu.kielce.pl/portal/Members/9487853b374349e88094142d8428c1f9/archiwum/2018-2019-semestr-letni/podstawy-programowania-2/pp2_instrukcja_9.pdf

https://pl.wikipedia.org/wiki/Binarne_drzewo_poszukiwa%C5%84

https://pl.wikipedia.org/wiki/Algorytm_DSW

https://eduinf.waw.pl/inf/alg/001_search/0116.php

https://www.geeksforgeeks.org/dsa/day-stout-warren-algorithm-to-balance-given-binary-search-tree/

https://www.geeksforgeeks.org/dsa/binary-search-tree-set-1-search-and-insertion/

https://www.geeksforgeeks.org/dsa/binary-search-tree-insert-parent-pointer/

https://www.pygame.org/docs/