# Raport Projektowy

Implementacja Drzewa BST ("Choinka")

Zuzanna Okońska, Aleksandra Syska

Grudzień 2025

**Temat 14 - Ubieramy choinkę**

**Streszczenie**

Niniejszy raport przedstawia proces projektowania i implementacji aplikacji w języku Python, modelującej strukturę Drzewa Poszukiwań Binarnych (BST) w kontekście tematycznym "Choinki". Dokument zawiera opis przyjętych struktur danych (w tym definicję łańcucha), analizę algorytmów, opis napotkanych problemów implementacyjnych oraz wyniki testów weryfikujących poprawność działania programu. Kompletny zapis działania programu wraz z wizualizacją drzew znajduje się w załączniku.

# Spis treści

**1 Wstęp i cel projektu** **3**

**2 Struktura danych** **3**
- 2.1 Definicja klasy węzła . . . 3
- 2.2 Definicja klasy choinki . . . 3
- 2.3 Definicja łańcucha . . . 3

**3 Opis algorytmów** **4**
- 3.1 Operacje podstawowe . . . 4
  - 3.1.1 Wstawianie elementu (insert) . . . 4
  - 3.1.2 Usuwanie elementu (delete) . . . 5
- 3.2 Algorytmy analityczne . . . 7
  - 3.2.1 Oświetlona (is_illuminated) . . . 7
  - 3.2.2 Równo Oświetlona (is_evenly_illuminated) . . . 8
  - 3.2.3 Stylowa (is_stylish) . . . 9
  - 3.2.4 Stabilna (is_stable) . . . 10
  - 3.2.5 NajdluzszyLancuchKolorowy (the_longest_colourful_path) . . . 11
  - 3.2.6 Elegancka (is_elegant) . . . 12
  - 3.2.7 Tradycyjna (is_traditional) . . . 14
  - 3.2.8 Gotowa(is_ready) . . . 15

**4 Opis użytkowania** **15**

**5 Testy i weryfikacja** **16**
5.1 Opis przypadków testowych . . . . . 16

**6 Napotkane błędy i ich rozwiązania** **17**

**7 Podział prac** **20**

**8 Wnioski** **20**
8.1 Bibliografia . . . . . 20

**A Wyniki testów i wizualizacja drzew** **21**

# 1 Wstęp i cel projektu

Celem projektu było stworzenie aplikacji, która implementuje strukturę **Drzewa Poszukiwań Binarnych (BST)** jako modelu "Choinki". Węzły drzewa reprezentują ozdoby choinkowe (bombki lub światełka) w zależności od wartości klucza liczbowego.

Projekt obejmuje implementację podstawowych operacji na drzewie (wstawianie, usuwanie, wyszukiwanie) oraz zaawansowanych algorytmów analitycznych sprawdzających specyficzne cechy choinki, takie jak oświetlenie, równomierne oświetlenie, stylowość, stabilność, elegancja, tradycyjność czy gotowość. Dodatkowo jest obsługiwane znalezienie długości najdłuższego łańcucha kolorowego.

Kod został napisany w języku Python na laptopach z oprogramowaniem Windows.

# 2 Struktura danych

Podstawowym elementem struktury jest klasa `Node`. Ze względu na optymalizację pamięciową, zastosowano mechanizm `__slots__`, ograniczający atrybuty obiektu do niezbędnego minimum.

## 2.1 Definicja klasy węzła

Każdy węzeł posiada:

- `key` (int) – unikalny klucz węzła (liczba naturalna).
- `left, right` – wskaźniki na poddrzewo lewe (w którym wartości kluczy węzłów są mniejsze od rozważanego węzła) i prawe poddrzewo (w którym wartości kluczy węzłów są większe od rozważanego węzła).

Dodatkowo zdefiniowano dynamiczne właściwości (Python `@property`):

- **Bombka:** Klucz jest liczbą nieparzystą ($key \pmod 2 \neq 0$).
- **Światełko:** Klucz jest liczbą parzystą ($key \pmod 2 = 0$).
- **Kolor światła:**
  - Żółte: podzielne przez 4 ($key \pmod 4 = 0$).
  - Czerwone: parzyste, niepodzielne przez 4.

## 2.2 Definicja klasy choinki

Każda choinka posiada węzeł korzeń (w drzewie pustym jest to None), od którego może odchodzić poddrzewo lewe i prawe poddrzewo.

## 2.3 Definicja łańcucha

W projekcie przyjęto specyficzną definicję **łańcucha** jako ścieżki w drzewie (ciągu węzłów połączonych krawędziami prowadzących od korzenia do liścia). W zależności od kontekstu (cechy choinki) wyróżniono:

- **Łańcuch kolorowy:** Ciąg węzłów składający się wyłącznie ze światełek. Bombki przerywają taki łańcuch.
- **Łańcuch monochromatyczny:** Ciąg węzłów o tym samym kolorze światła (np. tylko żółte), mogą być też bombki.
- **Łańcuch tradycyjny:** Ciąg węzłów taki, że liczba świateł żółtych jest równa liczbie świateł czerwonych, mogą być też bombki.

# 3 Opis algorytmów

W projekcie zastosowano podejście rekurencyjne, co jest naturalnym rozwiązaniem dla struktur drzewiastych. Kod został podzielony na metody publiczne (interfejs) oraz prywatne metody pomocnicze (zaczynające się od znaku `_`), które wykonują właściwe operacje na węzłach.

## 3.1 Operacje podstawowe

### 3.1.1 Wstawianie elementu (insert)

Metoda dodaje nowy węzeł do drzewa, zachowując porządek BST (mniejsze klucze na lewo, większe na prawo).
Jeśli klucz nie jest *int* lub jest mniejszy od 0, to funkcja pokazuje błąd. Jeśli węzeł o podanym kluczu już istnieje w drzewie, to funkcja nic nie zrobi.

**Metody pomocnicze:**

- `_insert_recursive(node, key)` – rekurencyjnie schodzi w głąb drzewa, szukając wolnego miejsca na nowy węzeł.
- `member(key)` – sprawdza, czy klucz już istnieje, aby zapobiec duplikatom.

**Algorytm 1** Rekurencyjne wstawianie węzła

```
1: function _INSERT_RECURSIVE(self, node, key)
2:     if key < node.key then                          ▷ Jeśli klucz mniejszy, idź w lewo
3:         if node.left is None then
4:             node.left ← NODE(key)                   ▷ Utwórz nowy węzeł w wolnym miejscu
5:         else
6:             SELF._INSERT_RECURSIVE(node.left, key)  ▷ Szukaj w lewym poddrzewie
7:         end if
8:     else                                            ▷ Jeśli klucz większy lub równy, idź w prawo
9:         if node.right is None then
10:            node.right ← NODE(key)                  ▷ Utwórz nowy węzeł w wolnym miejscu
11:        else
12:            SELF._INSERT_RECURSIVE(node.right, key)  ▷ Szukaj w prawym poddrzewie
13:        end if
14:    end if
15: end function
```

**Algorytm 2** Wyszukiwanie klucza

```
1: function MEMBER(self, key)
2:     curr ← self.root
3:     while curr is not None do
4:         if key = curr.key then
5:             return true
6:         else if key < curr.key then
7:             curr ← curr.left
8:         else
9:             curr ← curr.right
10:        end if
11:    end while
12:    return false
13: end function
```

**Metoda główna:**

**Algorytm 3** Wstawianie elementu

```
1: function INSERT(key)
2:     if key < 0 or key is not int then return      ▷ Walidacja: tylko liczby naturalne
3:     end if
4:     if MEMBER(key) then return                    ▷ Ignoruj duplikaty
5:     end if
6:     if root is None then
7:         root ← new Node(key)
8:     else
9:         INSERTRECURSIVE(root, key)
10:    end if
11: end function
```

### 3.1.2 Usuwanie elementu (delete)

Jest to najbardziej złożona operacja w strukturze BST, musząca obsłużyć trzy przypadki: usuwanie liścia, węzła z jednym dzieckiem oraz węzła z dwojgiem dzieci.

**Metody pomocnicze:**

- `_delete_recursive(node, key)` – główna logika przepinająca wskaźniki rodziców.
- `_get_min(node)` – kluczowa funkcja dla przypadku usuwania węzła z dwojgiem dzieci. Znajduje ona tzw. **następnika** (najmniejszy element w prawym poddrzewie), który zastępuje usuwany węzeł, zachowując spójność drzewa.

**Algorytm 4** Rekurencyjne usuwanie elementu

```
1: function _DeleteRecursive(node, key)
2:     if node is None then return None
3:     end if
4:     if key < node.key then
5:         node.left ← _DeleteRecursive(node.left, key)
6:     else if key > node.key then
7:         node.right ← _DeleteRecursive(node.right, key)
8:     else                                   ▷ Znaleziono węzeł do usunięcia
9:         if node.left is None then return node.right
10:        end if
11:        if node.right is None then return node.left
12:        end if
                                   ▷ Przypadek: węzeł ma dwoje dzieci
13:        succ ← _GetMin(node.right)                ▷ Znajdź następnika
14:        node.key ← succ.key                       ▷ Nadpisz wartość
15:        node.right ← DeleteRecursive(node.right, succ.key)
16:    end if
17:    return node
18: end function
```

**Algorytm 5** Znajdowanie najmniejszego węzła

```
1: function _get_min(self, node)
2:     while node.left is not None do
3:         node ← node.left
4:     end while
5:     return node
6: end function
```

**Metoda główna:**

**Algorytm 6** Usuwanie elementu

```
1: function DELETE(self, key)
2:     if key is not int or key < 0 then                ▷ Walidacja typu i wartości klucza
3:         print "Dozwolone są tylko liczby naturalne."
4:         return
5:     end if
6:     if self.root is None then                        ▷ Sprawdzenie czy drzewo nie jest puste
7:         print ("Nie ma nic do usunięcia (drzewo puste).")
8:         return
9:     end if
10:    if not SELF.MEMBER(key) then    ▷ Sprawdzenie czy element istnieje w drzewie
11:        print ("Element ", key, "nie istnieje w drzewie.")
12:        return
13:    end if
14:    self.root ← SELF._DELETE_RECURSIVE(self.root, key)        ▷ Wywołanie
   właściwego typu usuwania
15:    print ("Usunięto element ", key)
16: end function
```

## 3.2 Algorytmy analityczne

Poniżej przedstawiono algorytmy weryfikujące specyficzne cechy "Choinki". Każdy z nich opiera się na dedykowanych funkcjach zliczających.

### 3.2.1 Oświetlona (is_illuminated)

Drzewo jest oświetlone, jeśli każdy węzeł wewnętrzny (niebędący liściem) posiada w swoim poddrzewie przynajmniej jedno światełko.

**Metody pomocnicze:**

- `_count_lights(node)` – rekurencyjnie zlicza wszystkie węzły w poddrzewie, które są światełkami (klucz parzysty).
- `_check_illuminated(node)` - sprawdza, czy dany węzeł spełnia założenia

**Algorytm 7** Zliczanie podświetlonych węzłów

```
function _COUNT_LIGHTS(self, node)
    if node is None then                          ▷ Przypadek bazowy: koniec gałęzi
        return 0
    end if
    if node.is_light then                         ▷ Sprawdź czy bieżący węzeł świeci
        current_val ← 1
    else
        current_val ← 0
    end if
    return current_val + SELF._COUNT_LIGHTS(node.left) +
SELF._COUNT_LIGHTS(node.right)
end function
```

**Algorytm 8** Sprawdzanie, czy węzeł ma w poddrzewie światełko

```
1: function _CheckIlluminated(node)
2:     if node is None then return True
3:     end if
4:     if node jest liściem then return True
5:     end if
6:     Lights ← CountLights(node.left) + CountLights(node.right)
7:     if Lights == 0 then return False      ▷ Węzeł wewnętrzny bez świateł pod sobą
8:     end if
9:     return _CheckIlluminated(node.left) and _CheckIlluminated(node.right)
10: end function
```

**Metoda główna:**

**Algorytm 9** Sprawdzanie: Oświetlona

```
function is_illuminated(self)
    return self._check_illuminated(self.root)
end function
```

### 3.2.2 Równo Oświetlona (is_evenly_illuminated)

Drzewo musi być najpierw *Oświetlone*. Dodatkowo, dla każdego węzła różnica w liczbie świateł między lewym a prawym poddrzewem nie może być większa niż 1.

**Metody pomocnicze:**

- `_count_lights(node)` – używana do porównywania liczby świateł po obu stronach węzła. (opisana w poprzednim punkcie)
- `_check_evenly_illuminated(node)` - sprawdza, czy dany węzeł spełnia założenia

**Algorytm 10** Sprawdzanie równomiernego podświetlenia węzła

```
function _CheckEvenlyIlluminated(self, node)
    if node is None then                    ▷ Puste poddrzewo jest zrównoważone
        return true
    end if
    left_lights ← self._CountLights(node.left)
    right_lights ← self._CountLights(node.right)
    if abs(left_lights − right_lights) > 1 then      ▷ Sprawdź warunek równowagi
        return false
    end if
    return _CheckEvenlyIlluminated(node.left) and _CheckEvenlyIlluminated(node.right)
                                                    ▷ Rekurencyjne sprawdzenie dzieci
end function
```

**Metoda główna:**

**Algorytm 11** Sprawdzanie: Równo Oświetlona

```
1: function _IsEvenlyIlluminated(node)
2:     if not IsIlluminated(root) then return False
3:     end if
4:     return _CheckEvenlyIlluminated(root)
5: end function
```

### 3.2.3 Stylowa (is_stylish)

Choinka jest stylowa, jeśli żadna Bombka (liczba nieparzysta) nie ma w swoim poddrzewie Czerwonego Światełka.

**Metody pomocnicze:**

- `_has_red_light(node)` – sprawdza rekurencyjnie, czy w danym poddrzewie znajduje się jakikolwiek węzeł o kolorze "red".
- `is_bauble` – właściwość węzła (property), określająca czy jest on bombką.
- `_check_stylish(node)` - sprawdza, czy dany węzeł spełnia założenia

**Algorytm 12** Sprawdzanie obecności czerwonych świateł

```
function _HasRedLight(self, node)
    if node is None then
        return false
    end if
    if node.light_color = "red" then
        return true
    end if
    return self._HasRedLight(node.left) or self._HasRedLight(node.right)
end function
```

**Algorytm 13** Sprawdzanie, czy węzeł posiada potomków będących czerwonym światłem

```
1: function _CheckStylish(node)
2:     if node is None then return True
3:     end if
4:     if node.is_bauble then
5:         if _HasRedLight(node.left) or _HasRedLight(node.right) then return
   False                                  ▷ Naruszenie stylu: Bombka nad czerwonym
6:         end if
7:     end if
8:     return _CheckStylish(node.left) and _CheckStylish(node.right)
9: end function
```

**Metoda główna:**

**Algorytm 14** Sprawdzanie: Stylowa

```
function IsStylish(self)
    return self._CheckStylish(self.root)
end function
```

### 3.2.4 Stabilna (is_stable)

Drzewo jest stabilne, jeśli dla każdego węzła różnica liczby wszystkich potomków w lewym i prawym poddrzewie nie przekracza 2.

**Metody pomocnicze:**

- `_count_descendants(node)` – zlicza całkowitą liczbę węzłów w poddrzewie (zarówno bombek, jak i świateł).
- `_check_stable(node)` - sprawdza, czy dany węzeł spełnia założenia

**Algorytm 15** Zliczanie potomków węzła

```
function _CountDescendants(self, node)
    if node is None then
        return 0
    end if
    return 1 + self._CountDescendants(node.left) + self._CountDescendants(node.right)
end function
```

**Algorytm 16** Sprawdzanie: czy węzeł ma odpowiednią liczbę potomków w poddrzewach

```
1: function _CheckStable(node)
2:     if node is None then return True
3:     end if
4:     L ← _CountDescendants(node.left)
5:     R ← _CountDescendants(node.right)
6:     if |L − R| > 2 then return False
7:     end if
8:     return _CheckStable(node.left) and _CheckStable(node.right)
9: end function
```

**Metoda główna:**

**Algorytm 17** Sprawdzanie stabilności drzewa

```
function is_stable(self)
    return self._check_stable(self.root)
end function
```

### 3.2.5 NajdluzszyLancuchKolorowy (the_longest_colourful_path)

Zwraca długość najdłuższego łańcucha kolorowego.

**Metoda pomocnicza:**

- `_colourful_paths(self)` – funkcja wykorzystująca zmodyfikowaną wersję algorytmu DFS (Depth-First Search) zwraca listę kolorowych ścieżek. Gdy dojdzie do bombki, to nie idzie w głąb, tylko z pomocniczej listy `curr_path` usuwa ostatni węzeł i sprawdza, czy od ojca da się iść w inną stronę. Gdy zaś dojdzie do liścia będącego światełkiem, to dodaje łańcuch do listy `lancuchy`, usuwa z pomocniczej listy `curr_path` ostatni węzeł i również sprawdza inne drogi. Jeśli innych dróg nie ma, to znów usuwa ojca i tak rekurencyjnie, aż wszysktkie ścieżki zostanę sprawdzone.

**Algorytm 18** Wyznaczanie wszystkich kolorowych ścieżek

```
function _ColourfulPaths(self)
    if self.root is None then
        return []
    end if
    lancuchy ← []

    function _DFS(node, curr_path)
        curr_path.append(node.key)
        if node.left is None and node.right is None and node.is_light then
                    ▷ Jeśli doszliśmy do liścia, który jest światełkiem, to mamy łańcuch kolorowy, który dodajemy do listy
            lancuchy.append(LIST(curr_path))
        end if
                                    ▷ Idziemy dalej, tylko jeśli to światełko (lub korzeń)
        if node = self.root or node.is_light then
            if node.left is not None then
                _DFS(node.left, curr_path)
            end if
            if node.right is not None then
                _DFS(node.right, curr_path)
            end if
        end if
        curr_path.pop                    ▷ usuwamy ostatni węzeł z listy pomocniczej
    end function

    _DFS(self.root, [ ])
    return lancuchy
end function
```

**Metoda główna:**

**Algorytm 19** Wyznaczanie długości najdłuższej kolorowej ścieżki

```
function TheLongestColourfulPath(self)
    lancuchy ← self._ColourfulPaths        ▷ Pobranie listy wszystkich kolorowych ścieżek
    if not lancuchy then                   ▷ Sprawdzenie, czy lista ścieżek jest pusta
        return 0
    end if
    dlugosci_lancuchow ← []                ▷ Do tej listy będą dodawane kolejne długości kolorowych ścieżek
    for i ← 0 to len(lancuchy) − 1 do      ▷ Iteracja po wszystkich znalezionych ścieżkach
        dlugosci_lancuchow.append(len(lancuchy[i]))
    end for
    return max(dlugosci_lancuchow)         ▷ Zwrócenie największej znalezionej długości
end function
```

### 3.2.6 Elegancka (is_elegant)

Drzewo jest eleganckie, jeśli ma co najmniej tyle samo łańcuchów kolorowych, co jednobarwnych.

**Metody pomocnicze:**

- `_colourful_paths(self)` – opisana w poprzednim podpunkcie
- `_monochromatic_paths(self)` – funkcja wykorzystująca zmodyfikowaną wersję algorytmu DFS (Depth-First Search) zwraca listę monochromatycznych ścieżek. W odróżnieniu od funkcji `_colourful_paths()` bombki nie przerywają łańcucha, więc nie jest konieczne sprawdzanie, czy węzeł jest światełkiem. Funkcja zlicza liczbę żółtych i czerwonych światełek w danym łańcuchu i porównuje wyniki (przynajmniej jednego rodzaju światełek być nie może). Gdy funkcja dojdzie do liścia, to dodaje łańcuch do listy `lancuchy`, usuwa z pomocniczej listy `curr_path` ostatni węzeł i sprawdza, czy od ojca da się iść w inną stronę. Jeśli innych dróg nie ma, to znów usuwa ojca i tak rekurencyjnie, aż wszystkie ścieżki zostanę sprawdzone. Uwaga: drzewo z samych bombek też jest monochromatyczne.

**Algorytm 20** Wyznaczanie ścieżek monochromatycznych

```
function _MonochromaticPaths(self)
    if self.root is None then
        return []
    end if
    lancuchy ← []

    function _dfs(node, curr_path, n_of_yellow = 0, n_of_red = 0)
        curr_path.append(node.key)
        if node.light_color = 'yellow' then                ▷ Zliczamy konkretne węzły
            n_of_yellow ← n_of_yellow + 1
        end if
        if node.light_color = 'red' then
            n_of_red ← n_of_red + 1
        end if
                                          ▷ Monochromatyczny może zawierać bombki
        if node.left is None and node.right is None then
            if (n_of_red = 0 and n_of_yellow ≠ 0) or (n_of_yellow = 0 and n_of_red ≠ 0) then   ▷ Nie może zawierać światełek w dwóch kolorach
                lancuchy.append(list(curr_path))
            end if
        end if
        if node.left is not None then
            _dfs(node.left, curr_path, n_of_yellow, n_of_red)
        end if
        if node.right is not None then
            _dfs(node.right, curr_path, n_of_yellow, n_of_red)
        end if
        curr_path.pop                    ▷ Usuwamy ostatni węzeł z listy pomocniczej
    end function

    _dfs(self.root, [ ])
    return lancuchy
end function
```

**Metoda główna:**

**Algorytm 21** Sprawdzanie: Elegancka

```
1: function IsElegant(self)
2:     if len(self._ColourfulPaths(())) ≥ len(self._MonochromaticPaths(())) then
3:         return true
4:     else
5:         return false
6:     end if
7: end function
```

### 3.2.7 Tradycyjna (is_traditional)

Drzewo jest tradycyjne, jeśli jest stabilne, oświetlone i ma przynajmniej jeden łańcuch tradycyjny.

**Metoda pomocnicza:**

- `_traditional_paths(self)` – analogiczna do `_monochromatic_paths`, tylko liczba światełek żółtych i czerwonych musi być równa w danym łańcuchu. Uwaga: drzewo z samych bombek też jest tradycyjne.

**Algorytm 22** Wyznaczanie ścieżek tradycyjnych

```
function _TraditionalPaths(self)
    if self.root is None then
        return []
    end if
    lancuchy ← []

    function _DFS(node, curr_path, n_of_yellow = 0, n_of_red = 0)
        curr_path.append(node.key)
        if node.light_color = 'yellow' then                ▷ Zliczamy konkretne węzły
            n_of_yellow ← n_of_yellow + 1
        end if
        if node.light_color = 'red' then
            n_of_red ← n_of_red + 1
        end if
                                                ▷ Tradycyjny może zawierać bombki
        if node.left is None and node.right is None then
            if n_of_red = n_of_yellow then ▷ Światełek obu rodzajów musi być po równo
                lancuchy.append(LIST(curr_path))
            end if
        end if
        if node.left is not None then
            _DFS(node.left, curr_path, n_of_yellow, n_of_red)
        end if
        if node.right is not None then
            _DFS(node.right, curr_path, n_of_yellow, n_of_red)
        end if
        curr_path.pop                    ▷ Usuwamy ostatni węzeł z listy pomocniczej
    end function

    _DFS(self.root, [ ])
    return lancuchy
end function
```

**Metoda główna:**

**Algorytm 23** Sprawdzanie: Tradycyjna

```
1: procedure IsTraditional(self)
2:     if notself.is_stable() then
3:         return false
4:     end if
5:     if notself.is_illuminated() then
6:         return false
7:     end if
8:     if notself._TraditionalPaths() then
9:         return false
10:    end if
11:    return true
12: end procedure
```

### 3.2.8 Gotowa(is_ready)

Drzewo jest gotowe, jeśli jest stabilne, oświetlone i ma przynajmniej jeden łańcuch kolorowy i jeden łańcuch jednobarwny.

**Metody pomocnicze:** Wszystkie funkcje, które są wykorzystywane w sprawdzaniu, czy choinka jest gotowa, zostały opisane wcześniej.

**Metoda główna:**

**Algorytm 24** Sprawdzanie: Gotowa

```
1: function IsReady(self)
2:     if notself.is_stable() then
3:         return false
4:     end if
5:     if not self.IsIlluminated( )then
6:         return false
7:     end if
8:     if notself._ColourfulPaths() then
9:         return false
10:    end if
11:    if not self._MonochromaticPaths() then
12:        return false
13:    end if
14:    return true
15: end function
```

# 4 Opis użytkowania

Po wejściu w program użytkownik dostaje listę instrukcji obejmującej metody analityczne (oświetlona, itd) oraz klasyczne metody związane z drzewem BST (wstawianie, usuwanie, itd) pozwalające na jego edycję.

Dodatkowo jest opcja zobaczenia drzewa (nie jest to jednak klasyczny widok pionowy, w którym korzeń jest na górze, węzły o mniejszych kluczach po lewej, a węzły o większych kluczach po prawej, tylko implementacja programistyczna - obrócona o 90 stopni, w którym węzły o większych kluczach są na górze, korzeń mniej-więcej po środku, zaś węzły o mniejszych kluczach poniżej.)

Użytkownik jest proszony o wybranie numeru odpowiadającego metodzie z instrukcji. Każda instrukcja po poprawnym wywołaniu pokazuje odpowiedź, zaś przy błędnym wywołaniu wypisuje, na czym polega błąd.

# 5 Testy i weryfikacja

W celu potwierdzenia poprawności działania aplikacji przygotowano zestaw automatycznych scenariuszy testowych. Przetestowano 8 kluczowych funkcjonalności analitycznych drzewa.

**Uwaga:** Pełny zapis przebiegu wszystkich testów wraz z wizualizacją drzew znajduje się w **Załączniku A** na końcu niniejszego raportu.

## 5.1 Opis przypadków testowych

**Test 1: Oświetlona** Weryfikacja warunku obecności świateł pod węzłami wewnętrznymi.

- **Przypadek A (TRUE):** Drzewo Idealne (same światła). Algorytm potwierdził, że każdy węzeł wewnętrzny ma potomka będącego światełkiem.
- **Przypadek B (FALSE):** Drzewo, w którym jeden węzeł wewnętrzny posiada w swoim poddrzewie wyłącznie bombki.

**Test 2: Równo Oświetlona** Sprawdzono tolerancję różnicy świateł (limit = 1).

- **Przypadek A (TRUE):** Drzewo symetryczne (różnica 0).
- **Przypadek B (FALSE):** Drzewo asymetryczne (różnica świateł = 2, przy limicie 1).

**Test 3: Stylowa** Weryfikacja reguły: Bombka nie może mieć pod sobą czerwonego światła.

- **Przypadek A (TRUE):** Drzewo bez konfliktów bombka-czerwone światło.
- **Przypadek B (FALSE):** Drzewo zawierające ukryty błąd: węzeł 51 (bombka) posiada potomka 2 (czerwone światło).

**Test 4: Stabilna** Testowano zachowanie dla progu różnicy potomków równego 2.

- **Przypadek A (TRUE):** Drzewo zbalansowane.
- **Przypadek B (FALSE):** Lewa gałąź ma 10 elementów, prawa 13 (różnica 3, przy limicie 2).

**Test 5: Najdłuższy łańcuch kolorowy** Test przeprowadzono na drzewie o znanej strukturze. Algorytm poprawnie obliczył długość najdłuższego ciągu samych światełek.

**Test 6: Elegancka** Sprawdzenie relacji: liczba łańcuchów kolorowych $\geq$ liczba łańcuchów monochromatycznych.

- **Przypadek A (TRUE):** Drzewo bez bombek.
- **Przypadek B (FALSE):** Bombki przerywają łańcuchy kolorowe, zmniejszając ich liczbę poniżej liczby łańcuchów monochromatycznych.

**Test 7: Tradycyjna** Najbardziej złożony algorytm wymagający spełnienia wielu warunków (stabilność + oświetlenie + balans kolorów).

- **Przypadek A (TRUE):** Drzewo stabilne ze "złotą ścieżką"(balans kolorów 3 Żółte = 3 Czerwone).
- **Przypadek B (FALSE):** Drzewo bez czerwonych świateł (brak balansu).

**Test 8: Gotowa** Test agregujący wszystkie warunki końcowe.

- **Przypadek A (TRUE):** Drzewo spełniające wszystkie wymogi (jest stabilne, oświetlone i posiada odpowiednie łańcuchy).
- **Przypadek B (FALSE):** Drzewo niestabilne. Algorytm poprawnie zidentyfikował naruszenie podstawowego warunku strukturalnego i odrzucił drzewo jako "niegotowe".

# 6 Napotkane błędy i ich rozwiązania

- **Degeneracja drzewa przy losowaniu unikalnych wartości**
  Aby uniknąć duplikatów kluczy w losowym drzewie, wykorzystano strukturę zbioru (`set`). Ze względu na specyfikę języka Python, iteracja po zbiorze liczb całkowitych często zwracała je w kolejności rosnącej. Wstawianie posortowanych danych do niewyważonego BST skutkowało powstaniem drzewa zdegenerowanego (jednej długiej gałęzi), co uniemożliwiało poprawne testowanie. Problem rozwiązano poprzez konwersję zbioru na listę i jej wymieszanie funkcją `random.shuffle()` przed wstawieniem elementów do drzewa.

- **Implementacja łańcucha**
  Najpierw próbowałam przechodzić przez kolejne węzły łańcucha tak, jakby to była lista. Później próbowałam zaimplementować łańcuch jako dwukierunkową listę wiązaną, co było niepotrzebnym utrudnieniem. Ostatecznie skorzystałam ze zmodyfikowanej wersji algorytmu DFS (Depth-first search).

- **Błędy w trakcie pisania konkretnych funckji**

**Algorytm 25** Najdłuższa kolorowa ścieżka z interaktywnymi uwagami

```
function THE_LONGEST_COLOURFUL_PATH(self)
    if self.root is None then
        print "Drzewo jest puste, więc brak łańcuchów."
        return (1)
    end if
    lancuchy ← []

    function _DFS(node, curr_path)
        curr_path.append(node.key)
        if node.left is None and node.right is None and node.is_light then
            lancuchy.append(LIST(curr_path)) (2)
        end if
        if node = self.root or node.is_light then (3)
            if node.left is not None then
                _DFS(node.left, curr_path)
            end if
            if node.right is not None then
                _DFS(node.right, curr_path)
            end if
        end if(4)
        CURR_PATH.POP
    end function

    _DFS(self.root, [ ])
    if not lancuchy then
        return 0
    end if
    dlugosci_lancuchow ← []
    for i ← 0 to LEN(lancuchy) − 1 do
        dlugosci_lancuchow.append(LANCUCHY[I]()) (5)
    end for
    return len(max(dlugosci_lancuchow)) (6)
end function
```

**Lista uwag i poprawek:**
(1) tu powinno byc return [], by bylo spojne
(2) tu mialam curr_path.pop() i wtedy się za dużo usuwało, gdy bylo w dwoch miejscach, a z kolei, gdy bylo tylko tu, to za mało się usuwało
(3) wcześniej nie bylo dodanych warunkow o tym, że to powinno się robić, tylko, gdy jestesmy w korzeniu lub świetle
(4) gdy nie bylo warunkow, że potomek ma istniec, to potem byl blad, bo None nie ma klucza
(5) tu bez sensu jest tworzone jeszcze raz lancuchy; gdy zmienilam tu na len(lancuchy[i]), to wtedy w return trzeba usunac len, bo wtedy max(dlugosci_lancuchow) jest liczba
(6) wychodzilo zupelnie nie to, co trzeba

**Algorytm 26** Znajdowanie tradycyjnych ścieżek

```
function _TRADITIONAL_PATHS(self)
    if self.root is None then (1)
        print "Drzewo jest puste, więc brak łańcuchów."
        return []
    end if
    lancuchy ← []
(2)

    function _DFS(node, curr_path) (3)
        curr_path.append(node.key)
        if node = self.root then (4)
            n_of_red ← 0
            n_of_yellow ← 0
        end if
        if node ≠ self.root and node.light_color = 'yellow' then
            n_of_yellow ← n_of_yellow + 1
        end if
        if node ≠ self.root and node.light_color = 'red' then
            n_of_red ← n_of_red + 1
        end if
        if node.left is None and node.right is None and node.is_light and n_of_red =
n_of_yellow then (5)
            lancuchy.append(LIST(curr_path))
            n_of_red ← 0
            n_of_yellow ← 0
        end if
        if node = self.root or node.is_light then (6)
            if node.left is not None then
                _DFS(node.left, curr_path)
            end if
            if node.right is not None then
                _DFS(node.right, curr_path)
            end if
        end if
        CURR_PATH.POP
    end function

    _DFS(self.root, [ ])
    return lancuchy
end function
```

**Lista uwag i poprawek:**
(1) Wczesniej przypadek pustego drzewa był wewnątrz _dfs, wiec byl rozważany wielokrotnie
(2) Jest źle, bo za każdym razem, gdy program wchodzil w _dfs były zapominane poprzednie wartości n_of_red i n_of_yellow -> trzeba je dodać jako parametry funkcji _dfs
(3) Wcześniej bylo źle, bo definiowałam n_of_red = 0 i n_of_yellow = 0 przed _dfs
(4) Korzeń nie jest wliczany do zmiennych liczacych kolory
(5) node.is_light jest niepotrzebne
(6) To powinno się zawsze wykonywać, a nie tylko przy tym warunku

# 7 Podział prac

**Aleksandra Syska:**

Odpowiedzialna za architekturę klasy `Node` i `ChristmasTree`, implementację metod podstawowych (`insert`, `delete` z obsługą usuwania korzenia), walidację danych wejściowych, obsługę menu konsolowego oraz przygotowanie modułu testów automatycznych.

**Zuzanna Okońska:**

Opowiedzialna za implementację metody podstawowej `member`, implementację algorytmów analitycznych wykorzystujących DFS (szukanie łańcuchów kolorowych, monochromatycznych i tradycyjnych), logika metod `the_longest_colourful_path`, `is_traditional`, `is_elegant`, `is_ready` oraz implementacja metod zliczających właściwości węzłów. Dodatkowo odpowiedzialna za opisy (funkcji, napotkanych błędów oraz użytkowania) w raporcie.

# 8 Wnioski

Implementacja Drzewa BST w kontekście problemu ("Choinka") pozwoliła na głębsze zrozumienie rekurencji oraz algorytmów przeszukiwania drzewa.

Kluczowym wyzwaniem było prawidłowe obsłużenie usuwania węzłów, szukania odpowiednich łańcuchów oraz zaprojektowanie testów sprawdzających poprawność rozwiązania (w szczególności znalezienie choinki tradycyjnej było trudne). Zastosowanie `__slots__` w Pythonie okazało się dobrą praktyką optymalizacyjną.

## 8.1 Bibliografia


1. Prezentacje z kursu Algorytmy i Struktury danych na Politechnice Warszawskiej
2. Prezentacje z kursu Podstawy Programowania i Przetwarzania Danych na Politechnice Warszawskiej
3. https://eduinf.waw.pl/inf/utils/002_roz/mp001.php
4. https://eduinf.waw.pl/inf/alg/001_search/0109.php
5. https://eduinf.waw.pl/inf/alg/001_search/0114.php

# A Wyniki testów i wizualizacja drzew

Poniżej przedstawiono kompletny zrzut interfejsu oraz wyników testów automatycznych wygenerowany przez aplikację.

File - choinka_zuzia

```
C:\Users\olas0\anaconda3\python.exe C:\Users\olas0\PycharmProjects\pythonProject_choinka_ola\choinka_zuzia.py
Program uruchomiony. Choinka jest pusta.

==========================================
 SYSTEM OBSŁUGI CHOINKI 
==========================================
METODY ANALITYCZNE:
1. Sprawdź: Oświetlona
2. Sprawdź: Równo oświetlona
3. Sprawdź: Stylowa
4. Sprawdź: Stabilna
5. Sprawdź: Długość najdłuższego łańcucha kolorowego
6. Sprawdź: Elegancka
7. Sprawdź: Tradycyjna
8. Sprawdź: Gotowa
------------------------------------------
EDYCJA DRZEWA:
9. RESET (Wyczyść drzewo)
10. USUŃ element
11. WYLOSUJ nowe elementy (Dodaj 20 unikalnych liczb)
12. DODAJ RĘCZNIE (Wpisz własne liczby naturalne)
13. SPRAWDŹ, czy klucz jest w drzewie
------------------------------------------
INNE:
14. POKAŻ DRZEWO
15. GENERUJ RAPORT (Testy Automatyczne + Wizualizacja)
0. Wyjście

Wybierz opcję: 15

##################################################
       AUTOMATYCZNE TESTY DO RAPORTU
##################################################

======================================
1. Test cechy: OŚWIETLONA
```

```
CASE A (TRUE) [Drzewo Idealne - same światła] -> True

--- STRUKTURA CHOINKI ---
                  -> 124 [Światło (yellow)]
              -> 120 [Światło (yellow)]
                  -> 116 [Światło (yellow)]
          -> 112 [Światło (yellow)]
                  -> 108 [Światło (yellow)]
              -> 104 [Światło (yellow)]
                  -> 100 [Światło (yellow)]
      -> 96 [Światło (yellow)]
                  -> 92 [Światło (yellow)]
              -> 88 [Światło (yellow)]
                  -> 84 [Światło (yellow)]
          -> 80 [Światło (yellow)]
                  -> 76 [Światło (yellow)]
              -> 72 [Światło (yellow)]
                  -> 68 [Światło (yellow)]
-> 64 [Światło (yellow)]
                  -> 60 [Światło (yellow)]
              -> 56 [Światło (yellow)]
                  -> 52 [Światło (yellow)]
          -> 48 [Światło (yellow)]
                  -> 44 [Światło (yellow)]
              -> 40 [Światło (yellow)]
                  -> 36 [Światło (yellow)]
      -> 32 [Światło (yellow)]
                  -> 28 [Światło (yellow)]
              -> 24 [Światło (yellow)]
                  -> 20 [Światło (yellow)]
          -> 16 [Światło (yellow)]
                  -> 12 [Światło (yellow)]
              -> 8 [Światło (yellow)]
                  -> 4 [Światło (yellow)]
--------------------------

CASE B (FALSE) [Węzeł wewnętrzny (25) ma pod sobą
same bombki] -> False

--- STRUKTURA CHOINKI ---

```

```
        -> 214 [Światło (red)]

       -> 213 [Bombka]

   -> 212 [Światło (yellow)]

   211 [Bombka]

   Światło (red)]

   Bombka]

    (yellow)]

-------------------------

=======================================
2. Test cechy: RÓWNO OŚWIETLONA
CASE A (TRUE) [Idealna symetria] -> True

--- STRUKTURA CHOINKI ---

    -> 96 [Światło (yellow)]
```

```
                  -> 92 [Światło (yellow)]
             -> 88 [Światło (yellow)]
                  -> 84 [Światło (yellow)]
         -> 80 [Światło (yellow)]
                  -> 76 [Światło (yellow)]
             -> 72 [Światło (yellow)]
                  -> 68 [Światło (yellow)]
-> 64 [Światło (yellow)]
                  -> 60 [Światło (yellow)]
             -> 56 [Światło (yellow)]
                  -> 52 [Światło (yellow)]
         -> 48 [Światło (yellow)]
                  -> 44 [Światło (yellow)]
             -> 40 [Światło (yellow)]
                  -> 36 [Światło (yellow)]
    -> 32 [Światło (yellow)]
                  -> 28 [Światło (yellow)]
             -> 24 [Światło (yellow)]
                  -> 20 [Światło (yellow)]
         -> 16 [Światło (yellow)]
                  -> 12 [Światło (yellow)]
             -> 8 [Światło (yellow)]
                  -> 4 [Światło (yellow)]
-------------------------

CASE B (FALSE) [Różnica świateł = 2 (Limit=1)] ->
    False

--- STRUKTURA CHOINKI ---
                                                  ->
    185 [Bombka]
                                              -> 183 [
    Bombka]
                                         -> 181 [
    Bombka]
                                     -> 179 [Bombka]
                                 -> 177 [Bombka]
                             -> 175 [Bombka]
                         -> 160 [Światło (yellow)]
                     -> 158 [Światło (red)]
                 -> 156 [Światło (yellow)]
```

```
148              -> 154 [Światło (red)]
149          -> 152 [Światło (yellow)]
150      -> 150 [Światło (red)]
151 -> 100 [Światło (yellow)]
152      -> 50 [Światło (red)]
153          -> 25 [Bombka]
154                           -> 16 [Światło (yellow)]
155                       -> 12 [Światło (yellow)]
156                  -> 8 [Światło (yellow)]
157              -> 4 [Światło (yellow)]
158 ------------------------
159
160 ========================================
161 3. Test cechy: STYLOWA
162 CASE A (TRUE) [Brak bombek = stylowa] -> True
163
164 --- STRUKTURA CHOINKI ---
165                  -> 124 [Światło (yellow)]
166              -> 120 [Światło (yellow)]
167                  -> 116 [Światło (yellow)]
168          -> 112 [Światło (yellow)]
169                  -> 108 [Światło (yellow)]
170              -> 104 [Światło (yellow)]
171                  -> 100 [Światło (yellow)]
172      -> 96 [Światło (yellow)]
173                  -> 92 [Światło (yellow)]
174              -> 88 [Światło (yellow)]
175                  -> 84 [Światło (yellow)]
176          -> 80 [Światło (yellow)]
177                  -> 76 [Światło (yellow)]
178              -> 72 [Światło (yellow)]
179                  -> 68 [Światło (yellow)]
180 -> 64 [Światło (yellow)]
181                  -> 60 [Światło (yellow)]
182              -> 56 [Światło (yellow)]
183                  -> 52 [Światło (yellow)]
184          -> 48 [Światło (yellow)]
185                  -> 44 [Światło (yellow)]
186              -> 40 [Światło (yellow)]
187                  -> 36 [Światło (yellow)]
188      -> 32 [Światło (yellow)]
```

```
                  -> 28 [Światło (yellow)]
             -> 24 [Światło (yellow)]
                  -> 20 [Światło (yellow)]
         -> 16 [Światło (yellow)]
                  -> 12 [Światło (yellow)]
             -> 8 [Światło (yellow)]
                  -> 4 [Światło (yellow)]
-------------------------

CASE B (FALSE) [Ukryty błąd głęboko: Bombka(51) -> Czerwone(2)] -> False

--- STRUKTURA CHOINKI ---

        -> 609 [Bombka]

    -> 608 [Światło (yellow)]
                                                    -> 607 [Bombka]
                                                -> 606 [Światło (red)]
                                            -> 605 [Bombka]
                                        -> 604 [Światło (yellow)]
                                    -> 603 [Bombka]
                                -> 602 [Światło (red)]
                            -> 601 [Bombka]
                        -> 600 [Światło (yellow)]
                    -> 500 [Światło (yellow)]
             -> 400 [Światło (yellow)]
         -> 300 [Światło (yellow)]
     -> 200 [Światło (yellow)]
-> 100 [Światło (yellow)]
         -> 51 [Bombka]
     -> 50 [Światło (red)]
             -> 40 [Światło (yellow)]
         -> 25 [Bombka]
             -> 10 [Światło (red)]
                  -> 5 [Bombka]
                      -> 2 [Światło (red)]
```

```
------------------------

========================================
4. Test cechy: STABILNA
CASE A (TRUE) [Idealny balans] -> True

--- STRUKTURA CHOINKI ---
                    -> 124 [Światło (yellow)]
               -> 120 [Światło (yellow)]
                    -> 116 [Światło (yellow)]
          -> 112 [Światło (yellow)]
                    -> 108 [Światło (yellow)]
               -> 104 [Światło (yellow)]
                    -> 100 [Światło (yellow)]
     -> 96 [Światło (yellow)]
                    -> 92 [Światło (yellow)]
               -> 88 [Światło (yellow)]
                    -> 84 [Światło (yellow)]
          -> 80 [Światło (yellow)]
                    -> 76 [Światło (yellow)]
               -> 72 [Światło (yellow)]
                    -> 68 [Światło (yellow)]
-> 64 [Światło (yellow)]
                    -> 60 [Światło (yellow)]
               -> 56 [Światło (yellow)]
                    -> 52 [Światło (yellow)]
          -> 48 [Światło (yellow)]
                    -> 44 [Światło (yellow)]
               -> 40 [Światło (yellow)]
                    -> 36 [Światło (yellow)]
     -> 32 [Światło (yellow)]
                    -> 28 [Światło (yellow)]
               -> 24 [Światło (yellow)]
                    -> 20 [Światło (yellow)]
          -> 16 [Światło (yellow)]
                    -> 12 [Światło (yellow)]
               -> 8 [Światło (yellow)]
                    -> 4 [Światło (yellow)]
------------------------

CASE B (FALSE) [Różnica potomków = 3 (Limit=2)] ->
```

```
False

--- STRUKTURA CHOINKI ---

-> 230 [Światło (red)]
                                                    -> 220 [Światło (yellow)]
                                                 -> 210 [Światło (red)]
                                              -> 200 [Światło (yellow)]
                                           -> 190 [Światło (red)]
                                        -> 180 [Światło (yellow)]
                                     -> 170 [Światło (red)]
                                  -> 160 [Światło (yellow)]
                               -> 150 [Światło (red)]
                            -> 140 [Światło (yellow)]
                         -> 130 [Światło (red)]
                      -> 120 [Światło (yellow)]
                   -> 110 [Światło (red)]
-> 100 [Światło (yellow)]
                   -> 90 [Światło (red)]
                      -> 80 [Światło (yellow)]
                         -> 70 [Światło (red)]
                            -> 60 [Światło (yellow)]
                               -> 50 [Światło (red)]
                                  -> 40 [Światło (yellow)]
                                     -> 30 [Światło (red)]
                                        -> 20 [Światło (yellow)]
                                           -> 10 [Światło (red)]
                                              -> 5 [Bombka]
------------------------

========================================
5. Test: DŁUGOŚĆ ŁAŃCUCHA
Drzewo Idealne (wysokość 5). Wynik: 5
```

```
295
296 --- STRUKTURA CHOINKI ---
297                 -> 124 [Światło (yellow)]
298             -> 120 [Światło (yellow)]
299                 -> 116 [Światło (yellow)]
300         -> 112 [Światło (yellow)]
301                 -> 108 [Światło (yellow)]
302             -> 104 [Światło (yellow)]
303                 -> 100 [Światło (yellow)]
304     -> 96 [Światło (yellow)]
305                 -> 92 [Światło (yellow)]
306             -> 88 [Światło (yellow)]
307                 -> 84 [Światło (yellow)]
308         -> 80 [Światło (yellow)]
309                 -> 76 [Światło (yellow)]
310             -> 72 [Światło (yellow)]
311                 -> 68 [Światło (yellow)]
312 -> 64 [Światło (yellow)]
313                 -> 60 [Światło (yellow)]
314             -> 56 [Światło (yellow)]
315                 -> 52 [Światło (yellow)]
316         -> 48 [Światło (yellow)]
317                 -> 44 [Światło (yellow)]
318             -> 40 [Światło (yellow)]
319                 -> 36 [Światło (yellow)]
320     -> 32 [Światło (yellow)]
321                 -> 28 [Światło (yellow)]
322             -> 24 [Światło (yellow)]
323                 -> 20 [Światło (yellow)]
324         -> 16 [Światło (yellow)]
325                 -> 12 [Światło (yellow)]
326             -> 8 [Światło (yellow)]
327                 -> 4 [Światło (yellow)]
328 -------------------------
329
330 ========================================
331 6. Test cechy: ELEGANCKA (Kolorowe >= Mono)
332 CASE A (TRUE) [Bez bombek: Kolorowe == Mono] -> True
333
334 --- STRUKTURA CHOINKI ---
335                 -> 124 [Światło (yellow)]
```

```
336              -> 120 [Światło (yellow)]
337                  -> 116 [Światło (yellow)]
338          -> 112 [Światło (yellow)]
339                  -> 108 [Światło (yellow)]
340              -> 104 [Światło (yellow)]
341                  -> 100 [Światło (yellow)]
342      -> 96 [Światło (yellow)]
343                  -> 92 [Światło (yellow)]
344              -> 88 [Światło (yellow)]
345                  -> 84 [Światło (yellow)]
346          -> 80 [Światło (yellow)]
347                  -> 76 [Światło (yellow)]
348              -> 72 [Światło (yellow)]
349                  -> 68 [Światło (yellow)]
350 -> 64 [Światło (yellow)]
351                  -> 60 [Światło (yellow)]
352              -> 56 [Światło (yellow)]
353                  -> 52 [Światło (yellow)]
354          -> 48 [Światło (yellow)]
355                  -> 44 [Światło (yellow)]
356              -> 40 [Światło (yellow)]
357                  -> 36 [Światło (yellow)]
358      -> 32 [Światło (yellow)]
359                  -> 28 [Światło (yellow)]
360              -> 24 [Światło (yellow)]
361                  -> 20 [Światło (yellow)]
362          -> 16 [Światło (yellow)]
363                  -> 12 [Światło (yellow)]
364              -> 8 [Światło (yellow)]
365                  -> 4 [Światło (yellow)]
366 -------------------------
367
368 CASE B (FALSE) [Bombki blokują łańcuchy Kolorowe,
    ale nie Mono] -> False
369
370 --- STRUKTURA CHOINKI ---
371
                             -> 329 [Bombka]
372
                         -> 327 [Bombka]
373
```

```
                  -> 325 [Bombka]

               -> 323 [Bombka]

            -> 321 [Bombka]

        -> 319 [Bombka]

    -> 317 [Bombka]
                                                    ->
    315 [Bombka]
                                                 -> 313 [
    Bombka]
                                              -> 311 [
    Bombka]
                                         -> 309 [Bombka]
                                     -> 307 [Bombka]
                                 -> 305 [Bombka]
                             -> 303 [Bombka]
                         -> 301 [Bombka]
                  -> 204 [Światło (yellow)]
              -> 201 [Bombka]
          -> 152 [Światło (yellow)]
     -> 151 [Bombka]
-> 100 [Światło (yellow)]
     -> 51 [Bombka]
          -> 4 [Światło (yellow)]
-------------------------

=======================================
7. Test cechy: TRADYCYJNA
CASE A (TRUE) [Istnieje ścieżka z równą liczbą Y i R
    ] -> True

--- STRUKTURA CHOINKI ---
                         -> 194 [Światło (red)]
                  -> 192 [Światło (yellow)]
              -> 190 [Światło (red)]
          -> 176 [Światło (yellow)]
                  -> 164 [Światło (yellow)]
              -> 162 [Światło (red)]
```

```
406     -> 150 [Światło (red)]
407             -> 138 [Światło (red)]
408                 -> 130 [Światło (red)]
409         -> 124 [Światło (yellow)]
410             -> 110 [Światło (red)]
411 -> 100 [Światło (yellow)]
412             -> 90 [Światło (red)]
413                 -> 80 [Światło (yellow)]
414         -> 76 [Światło (yellow)]
415             -> 62 [Światło (red)]
416     -> 50 [Światło (red)]
417             -> 38 [Światło (red)]
418                 -> 36 [Światło (yellow)]
419         -> 24 [Światło (yellow)]
420             -> 10 [Światło (red)]
421                 -> 4 [Światło (yellow)]
422                     -> 2 [Światło (red)]
423 -------------------------
424
425 CASE B (FALSE) [Brak czerwonych świateł -> brak
    równowagi] -> False
426
427 --- STRUKTURA CHOINKI ---
428                 -> 124 [Światło (yellow)]
429             -> 120 [Światło (yellow)]
430                 -> 116 [Światło (yellow)]
431         -> 112 [Światło (yellow)]
432                 -> 108 [Światło (yellow)]
433             -> 104 [Światło (yellow)]
434                 -> 100 [Światło (yellow)]
435     -> 96 [Światło (yellow)]
436                 -> 92 [Światło (yellow)]
437             -> 88 [Światło (yellow)]
438                 -> 84 [Światło (yellow)]
439         -> 80 [Światło (yellow)]
440                 -> 76 [Światło (yellow)]
441             -> 72 [Światło (yellow)]
442                 -> 68 [Światło (yellow)]
443 -> 64 [Światło (yellow)]
444                 -> 60 [Światło (yellow)]
445             -> 56 [Światło (yellow)]
```

```
                    -> 52 [Światło (yellow)]
          -> 48 [Światło (yellow)]
                    -> 44 [Światło (yellow)]
              -> 40 [Światło (yellow)]
                    -> 36 [Światło (yellow)]
     -> 32 [Światło (yellow)]
                    -> 28 [Światło (yellow)]
              -> 24 [Światło (yellow)]
                    -> 20 [Światło (yellow)]
          -> 16 [Światło (yellow)]
                    -> 12 [Światło (yellow)]
              -> 8 [Światło (yellow)]
                    -> 4 [Światło (yellow)]
-------------------------

=======================================
8. Test cechy: GOTOWA
CASE A (TRUE) [Spełnia wszystkie warunki] -> True

--- STRUKTURA CHOINKI ---
                    -> 124 [Światło (yellow)]
              -> 120 [Światło (yellow)]
                    -> 116 [Światło (yellow)]
          -> 112 [Światło (yellow)]
                    -> 108 [Światło (yellow)]
              -> 104 [Światło (yellow)]
                    -> 100 [Światło (yellow)]
     -> 96 [Światło (yellow)]
                    -> 92 [Światło (yellow)]
              -> 88 [Światło (yellow)]
                    -> 84 [Światło (yellow)]
          -> 80 [Światło (yellow)]
                    -> 76 [Światło (yellow)]
              -> 72 [Światło (yellow)]
                    -> 68 [Światło (yellow)]
-> 64 [Światło (yellow)]
                    -> 60 [Światło (yellow)]
              -> 56 [Światło (yellow)]
                    -> 52 [Światło (yellow)]
          -> 48 [Światło (yellow)]
                    -> 44 [Światło (yellow)]
```

```
487              -> 40 [Światło (yellow)]
488                  -> 36 [Światło (yellow)]
489     -> 32 [Światło (yellow)]
490                  -> 28 [Światło (yellow)]
491              -> 24 [Światło (yellow)]
492                  -> 20 [Światło (yellow)]
493         -> 16 [Światło (yellow)]
494                  -> 12 [Światło (yellow)]
495              -> 8 [Światło (yellow)]
496                  -> 4 [Światło (yellow)]
497 ------------------------
498
499 CASE B (FALSE) [Niestabilna, więc niegotowa] ->
    False
500
501 --- STRUKTURA CHOINKI ---
502
    -> 230 [Światło (red)]
503                                                  ->
    220 [Światło (yellow)]
504                                                -> 210 [
    Światło (red)]
505                                             -> 200 [
    Światło (yellow)]
506                                          -> 190 [Światło
     (red)]
507                                       -> 180 [Światło (
    yellow)]
508                                  -> 170 [Światło (red)]
509                             -> 160 [Światło (yellow)]
510                        -> 150 [Światło (red)]
511                  -> 140 [Światło (yellow)]
512              -> 130 [Światło (red)]
513         -> 120 [Światło (yellow)]
514     -> 110 [Światło (red)]
515 -> 100 [Światło (yellow)]
516     -> 90 [Światło (red)]
517         -> 80 [Światło (yellow)]
518              -> 70 [Światło (red)]
519                  -> 60 [Światło (yellow)]
520                      -> 50 [Światło (red)]
```

File - choinka_zuzia

```
                            -> 40 [Światło (yellow)]
                                -> 30 [Światło (red)]
                                    -> 20 [Światło (
    yellow)]
                                        -> 10 [Światło (
    red)]
                                            -> 5 [Bombka
    ]
 -------------------------

 ##################################################
    ########
 Naciśnij ENTER, aby wrócić do menu...
```