Spis treści

Drzewa BST

Projekt II na Algorytmy i Struktury danych

Kacper Sołtysiak

416853

Omówienie projektu

Celem projektu było zbadanie właściwości drzew binarnych BST (Binary Search Tree) tworzonych z dużych zbiorów danych. Analizowano strukturę drzewa powstałego po losowym dodaniu 10 000 elementów. Eksperyment powtórzono 300 razy, aby uzyskać bardziej wiarygodne wyniki statystyk.

- Analizę wysokości drzewa po wstawieniu elementów w losowej kolejności – czyli maksymalną głębokość węzła, liczona jako liczba krawędzi od korzenia do najdalszego liścia.
- Pomiar liczby liści, czyli węzłów nieposiadających dzieci. (nie idą ani w lewo, ani w prawo)
- Wyznaczenie średniej głębokości węzłów drzewa – przeciętna odległość wszystkich węzłów od korzenia.

Drzewo tworzono na bazie listy liczb od 0 do 9999, która następnie była losowo mieszana za pomocą algorytmu Fisher-Yates. Dzięki temu uzyskiwano za każdym razem inną permutację, w której każdy możliwy układ miał takie samo prawdopodobieństwo. Po przetasowaniu elementy były po kolei dodawane do pustego drzewa BST zgodnie z zasadą: „lewo < rodzic < prawo”

Dodatkowo dla wysokości drzewa obliczono:

- Odchylenie standardowe – klasyczną miarę rozrzutu wyników wokół średniej.
- Rozstęp międzykwartylowy (IQR) – czyli różnicę między trzecim i pierwszym kwartylem, co pozwala ocenić „gęstość” danych bez wpływu wartości odstających.

Dla każdej pary badanych cech obliczono również korelację, aby sprawdzić zależność np. pomiędzy wysokością a liczbą liści, albo głębokością a liczbą węzłów.

Wyliczone dla 300 prób dane, zapisane zostały do pliku tekstowego. Dodatkowo, dane przedstawiono jako wykresy słupkowe interaktywne w matplotlib. W projekcie zastosowano również pare wykresów zrobionych „ręcznie” w klasycznym Excelu.

Omówienie kodu

```
7   def losuj(lista):                                   #Fisher-Yates shuffle
8       for i in range(len(lista) - 1, 0, -1):          #iteracja po "liscie" od ostatniego elementu
9           j = random.randint(0, i)                    #losowanie j z przedzialu 0,i włącznie
10          lista[i], lista[j] = lista[j], lista[i]     #zamiana miejsc i z j
```

Ta funkcja realizuje algorytm Fisher-Yates Shuffle, który służy do losowego przetasowania listy w sposób całkowicie losowy i uczciwy. Dzięki temu każda możliwa permutacja elementów ma taką samą szansę na wystąpienie, co jest kluczowe przy badaniach statystycznych.

Klasy

```
12  class Node:
13      def __init__(self, v):
14          self.value = v                              # przechowanie wartości
15          self.left = None                            # wskaźnik lewego potomka
16          self.right = None                           # wskaźnik prawego potomka
17
18  class BST:
19      def __init__(self):
20          self.root = None                            #korzeń drzewa is null, pusty
21
22      def dodaj(self, v):                             #dodawanie nowej wartosci do drzewa
23          if self.root is None:                       #jesli jest puste
24              self.root = Node(v)                     #nowy korzen
25          else:
26              self._dodaj(self.root, v)               # a jesli nie, wywoalnie rekurencyjnego wstawiania
27
28      def _dodaj(self, node, v):                      # rekurencyjne wstawianie
29          if v < node.value:                          # jesli wartosc mniejsza, idziemy w lewo
30              if node.left is None:
31                  node.left = Node(v)
32              else:
33                  self._dodaj(node.left, v)           # Jeśli znajdziemy wolne miejsce, tworzymy nowy węzeł
34          elif v > node.value:                        # jesli wieksza, to w prawo
35              if node.right is None:
36                  node.right = Node(v)
37              else:
38                  self._dodaj(node.right, v)          # # Jeśli znajdziemy wolne miejsce, tworzymy nowy węzeł
```

Class node, prosta klasa opisująca drzewa BST. Służy jako element składowy drzewa.
Class BST, klasa tworząca całę drzewo binarne. Tworzy drzewo z pustym korzeniem,

Funkcja „dodaj”, dodaje nową wartość v do drzewa, jeśli korzeń jest pusty – tworzy nowy węzeł, jeśli nie – wywołuje funkcje „_dodaj”

Funkcja „_dodaj” – rekurencyjna funkcja wstawiająca element. Jeśli znajdzie wolne miejsce – tworzy nowy węzeł, jeśli nie – idzie w głąb drzewa.

Funkcje

```
40      def wysokosc(self):                                 # obliczanie wysokosci drzewa
41          def f(n):
42              if n is None:
43                  return -1                               # jesli napotkamy puste drzewo, zwraca -1
44              return 1 + max(f(n.left), f(n.right))       # sprawdzamy wys. lewej i prawej galęzi, zwraca większą z nich +1
45          return f(self.root)
46
47      def liscie(self):                                   # liczenie liczby liści
48          def f(n):                                       # lisc to węzeł  bez dzieci
49              if n is None:
50                  return 0
51              if n.left is None and n.right is None:      # jeśli oba "dzieci" puste - zwraca 1
52                  return 1
53              return f(n.left) + f(n.right)               # w przeciwnym razie, sumujemy liście z prawej i lewej
54          return f(self.root)
55
56      def srednia_glebokosc(self):                        # srednia glebokosc wezłow
57          def f(n, g):                                    # g - aktualna glebokosc
58              if n is None:                               # jesli puste, zwracamy 0
59                  return (0, 0)
60              l_sum, l_ile = f(n.left, g + 1)             # suma glebokosci, liczba wezlow z lewej
61              r_sum, r_ile = f(n.right, g + 1)            # suma glebokosci, liczba wezlow z prawej
62              return (l_sum + r_sum + g, l_ile + r_ile + 1) # dodajemy głębokości po przejściu przez lewe i prawe
63          suma, ile = f(self.root, 0)
64          return suma / ile if ile > 0 else 0             # sumę głębkości dzielimy przez liczbę węzłów
```

Wysokość

Funkcja wysokosc – oblicza maksymalną wysokość drzewa, czyli najdłuższą ścieżkę od korzenia do liścia. Działa rekurencyjnie, szukając większej wartości spośród lewej i prawej strony drzewa i dodając

Liście

Funkcja liscie – zwraca liczbę liści w drzewie, czyli węzłów nieposiadających dzieci. Sprawdza każdy węzeł i jeśli nie ma potomków, zalicza go jako liść.

Średnia głębokość

Funkcja srednia_glebokosc – liczy średnią głębokość wszystkich węzłów w drzewie. Podczas rekurencji zlicza sumę głębokości oraz ilość węzłów, a następnie dzieli sumę przez liczbę.

Funkcje statystyczne

```python
def std(dane):                                  # funkcja obliczająca odchylenie standardowe
    s = sum(dane) / len(dane)                   # średnia z danych
    return math.sqrt(sum((x - s) ** 2 for x in dane) / len(dane))      # wzor na odchylenie za pomocą "import math"

def korelacja(x, y):                            # funkcja licząca korelacje Pearsona między dwiema listami
    n = len(x)
    sx = sum(x) / n                             # średnia z X
    sy = sum(y) / n                             # średnia z Y
    licznik = sum((x[i] - sx)*(y[i] - sy) for i in range(n))    # suma iloczynów odchyleń
    mianownik = math.sqrt(sum((x[i] - sx)**2 for i in range(n)) * sum((y[i] - sy)**2 for i in range(n)))    # pierwiastek z iloczynu sum kwadratów
    return licznik / mianownik if mianownik != 0 else 0     # zwróć wynik lub 0, jeśli mianownik = 0

def iqr(dane):                                  # funkcja licząca odstęp międzykwartylowy
    dane_posortowane = posortowane(dane)        # posortowana lista
    n = len(dane_posortowane)
    q1_index = n // 4                           # indeks I kwartylu
    q3_index = (3 * n) // 4                     # indeks III kwartylu
    return dane_posortowane[q3_index] - dane_posortowane[q1_index]  # różnica Q3 - Q1
```

Funkcja std liczy odchylenie standardowe, czyli jak bardzo dane są rozproszone wokół średniej. Funkcja korelacja oblicza korelację Pearsona pomiędzy dwiema listami – sprawdza, czy wartości w jednej liście rosną razem z drugą. Funkcja iqr oblicza rozstęp międzykwartylowy, czyli różnicę między 3. a 1. kwartylem – dobra miara rozrzutu danych.

Główny blok programu

```python
if __name__ == "__main__":                      # Główna część programu, uruchamiana tylko bezpośrednio
    h, l, d = [], [], []                        # Listy: wysokości, liści, głębkości
    for p in range(300):                        # 300 prób
        dane = list(range(10000))               # Lista 0-9999
        losuj(dane)                             # pomieszanie listy
        drzewo = BST()                          # tworzymy nowe drzewo BST
        for x in dane:                          # dodajemy wszystkie wartości do drzewa
            drzewo.dodaj(x)                     # dodajemy dane do drzewa
        h.append(drzewo.wysokosc())             # zapisanie wysokości
        l.append(drzewo.liscie())               # zapisanie liczby liści
        d.append(drzewo.srednia_glebokosc())    # zapisanie średniej głębokości
```

Blok if __name__ == "__main__": to główna pętla programu. Wykonuje 300 powtórzeń doświadczenia. W każdej próbie:

- tworzona jest lista liczb 0–9999,
- dane są losowo mieszane (shuffle),
- budowane jest nowe drzewo BST,
- dodawane są do niego wszystkie liczby,
- a następnie mierzone są trzy wartości:
    - wysokość drzewa,
    - liczba liści,
    - średnia głębokość węzłów.

Wszystkie wyniki są zapisywane do osobnych list, które potem posłużą do wykresów i analizy.

Zapis wyników

```
    # Zapis wyników do pliku tekstowego
    with open("wyniki_bst.txt", "w") as f:                    # tworzy .txt w trybie zapisu "write"
        for i in range(len(h)):                               # iteruje po wszystkich zebranych wynikach
            f.write(f"Próba {i+1}: Wysokość = {h[i]}, Liście = {l[i]}, Średnia głębokość = {round(d[i], 3)}\n")
            # zapisuje jedna linie z wynikami; wysokość, liście, średnia gł.
```

Ten fragment zapisuje wyniki eksperymentu do pliku tekstowego wyniki_bst.txt. Dla każdej z 300 prób zapisuje jedną linię w formacie:

Próba 1: Wysokość = ..., Liście = ..., Średnia głębokość = ...

Dzięki temu można później otworzyć plik w notatniku, wydrukować go lub wkleić do prezentacji jako dodatkowy materiał dowodowy. Jest to bardzo prosta, ale przydatna forma archiwizacji danych.

Printy statystyk

```
    print("\nKorelacje:")
    print("wysokosc vs liscie:", round(korelacja(h, l), 3))         # korelacja między wysokością a liczbą liści
    print("wysokosc vs glebokosc:", round(korelacja(h, d), 3))      # korelacja między wysokością a głębokościa
    print("liscie vs glebokosc:", round(korelacja(l, d), 3))        # korelacja między liści a głębokością

    print("\nOdchylenie standardowe wysokosci:", round(std(h), 3))  # odchylenie st. wysokości drzewa
    print("Rozstęp międzykwartylowy wysokości:", iqr(h))            # rozstęp międzykwartylowy wysokości

```

Ten fragment kodu wyświetla w konsoli najważniejsze statystyki podsumowujące projekt. Wypisuje:

- Korelacje Pearsona między wszystkimi trzema zmiennymi: wysokością, liczbą liści i średnią głębokością. Dzięki temu możemy sprawdzić, czy np. wyższe drzewa mają więcej liści, albo czy są głębsze.
- Odchylenie standardowe wysokości, które pokazuje, jak bardzo wysokości drzew różnią się od średniej.
- Rozstęp międzykwartylowy (IQR), który mierzy "rozciągnięcie" danych pomiędzy kwartylami Q1 i Q3 — czyli ile wynosi szerokość środkowych 50% wyników.

Wszystkie wyniki są zaokrąglone do 3 miejsc po przecinku, żeby były czytelne i gotowe do umieszczenia w raporcie lub prezentacji.

Funkcje tworzące wykresy matplotlib

```python
# Tworzenie i zapisywanie wykresów histogramów
def zapisz_wykresy(heights, leaves, depths):  # Funkcja przyjmuje 3 listy z wynikami

    # Wysokość drzewa
    plt.figure()  # Tworzy nową figurę (okno wykresu)
    plt.hist(heights, bins=20, color='skyblue', edgecolor='black')  # Rysuje histogram z 20 przedziałami
    plt.title("Histogram wysokości drzewa")  # Tytuł wykresu
    plt.xlabel("Wysokość")  # Opis osi X
    plt.ylabel("Liczba przypadków")  # Opis osi Y
    plt.show()  # Wyświetlenie wykresu

    # Liczba liści
    plt.figure()
    plt.hist(leaves, bins=20, color='lightgreen', edgecolor='black')
    plt.title("Histogram liczby liści")
    plt.xlabel("Liście")
    plt.ylabel("Liczba przypadków")
    plt.show()

    # Średnia głębokość
    plt.figure()
    plt.hist(depths, bins=20, color='salmon', edgecolor='black')
    plt.title("Histogram średniej głębokości")
    plt.xlabel("Średnia głębokość")
    plt.ylabel("Liczba przypadków")
    plt.show()
```

Blok ten odpowiada za wyświetlenie trzech wykresów (histogramów) interaktywnych, po jednym dla każdej mierzonej cechy; wysokości, liczby liści, średniej głębokości.

Dla każdej z cech tworzy osobne okno wykresu (plt.figure()), rysuje histogram z 20 przedziałami (bins=20), ustawia tytuł, etykiety osi oraz kolory słupków.

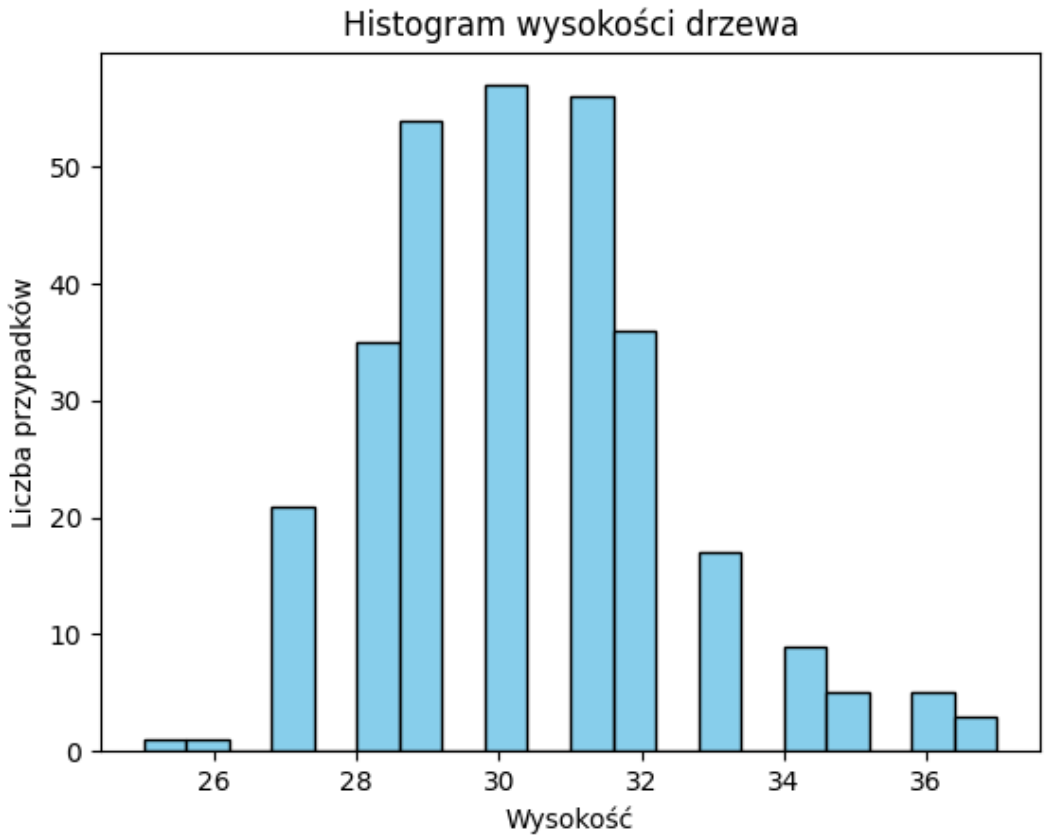
Wykresy

Wysokość drzewa

Wykres przedstawia liczbę drzew o danej wysokości, zmierzoną w 300 próbach.

Wnioski:

- Domimanta to wartość 30.
- Rozkład z lekka prawoskośny, niewielka liczba drzew o dużej wysokości
- Drzewa BST nie są zbalansowane, przy losowym wstawianiu.
- Dane skupione wokół środka – potwierdza to też niskie odchylenie standardowe i IQR\

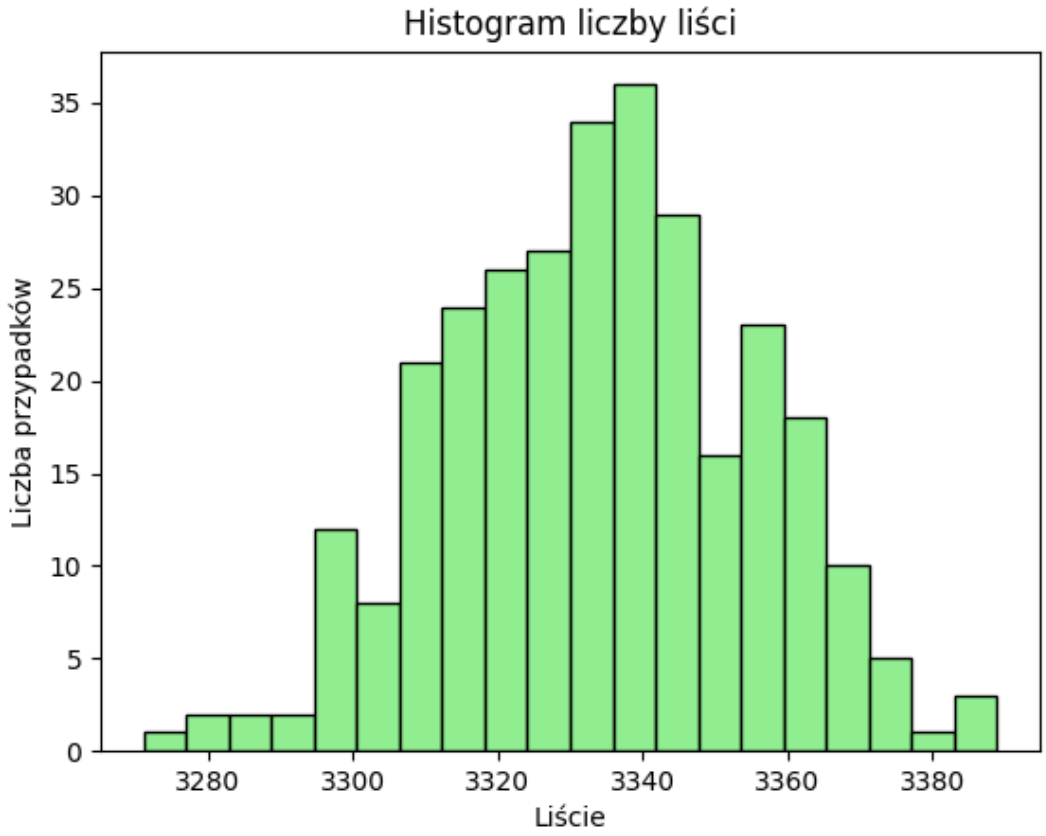


Liczba liści

Wykres pokazuje rozkład liczby liści w drzewach BST w każdej z prób.

Wnioski:

- Typowa liczba liści wynosi około 3340.
- Rozkład jest symetryczny i zbliżony do normalnego (kształt dzwonu).
- Sugeruje to, że liczba liści w losowo generowanych drzewach BST nie zmienia się drastycznie.
- Im więcej liści, tym więcej węzłów „kończących gałęzie”, co oznacza dobrą rozbudowę struktury.



Średnia głębokość

Wykres przedstawia, jak głęboko przeciętnie znajdują się węzły w drzewie.

Wnioski:

- Średnia głębokość najczęściej mieści się w zakresie 15–16.
- Rozkład jest asymetryczny, z wyraźnym ogonem w prawo — kilka przypadków głębokich drzew wpływa na średnią.
- Mniejsza głębokość oznacza krótsze ścieżki do danych, co z punktu widzenia algorytmów wyszukiwania jest korzystne.
- Im większa wysokość drzewa, tym wyższa średnia głębokość — co potwierdza obserwowaną korelację

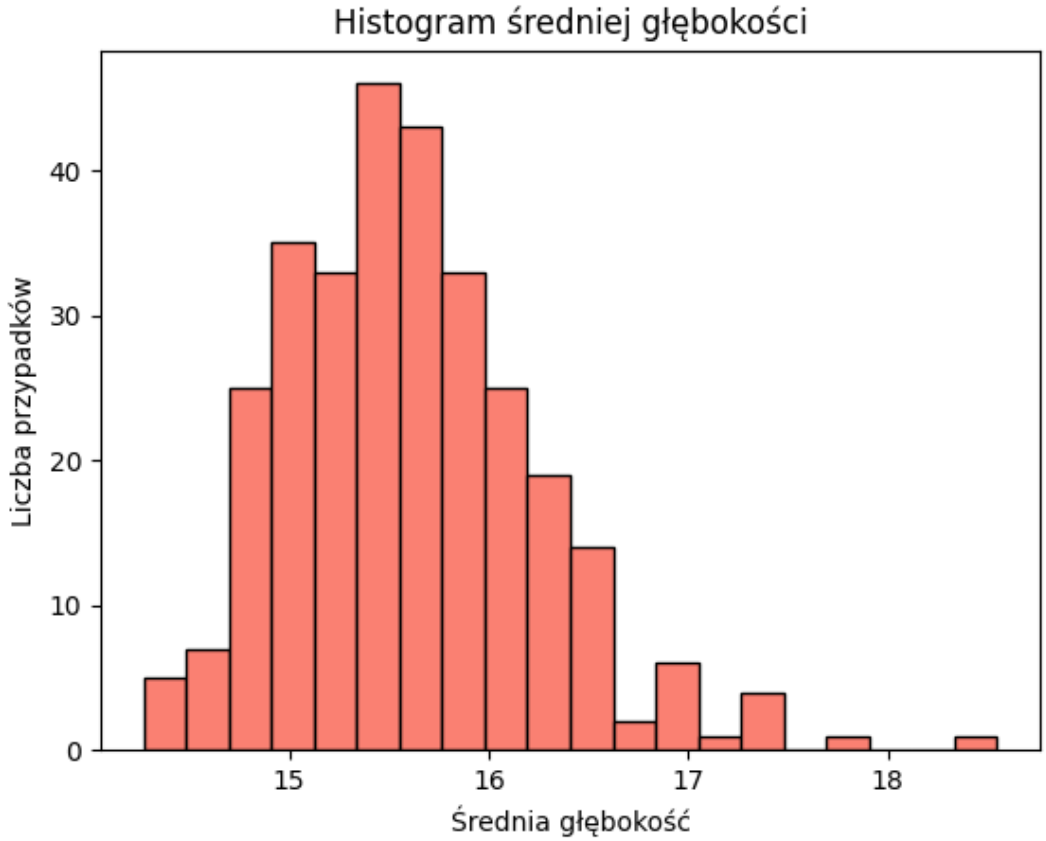


Tabela statystyczna

	Średnia	Mediana	Odch. standardowe	IQR	Min	Max
Wysokość	27,017	27	2,067	4	24	30
Liście	3348,66	3348	29,175	52,25	3300	3400
Średnia głębokość	37,445	37,307	1,461	2,538	35,001	39,979

Podsumowanie projektu

W ramach projektu zbadano strukturę drzewa BST tworzonego na podstawie losowych danych. W 300 powtórzeniach analizowano trzy kluczowe cechy drzewa: wysokość, liczbę liści oraz średnią głębokość węzłów.

Wyniki pokazały, że drzewa BST tworzone z losowych permutacji nie są optymalnie zbalansowane, ale zachowują umiarkowaną regularność. Wysokość drzew mieściła się zazwyczaj w przedziale 27–32 poziomów, liczba liści wynosiła około 3300–3400, a średnia głębokość węzłów oscylowała wokół 35–37.

Dodatkowo obliczono odchylenie standardowe, rozstęp międzykwartylowy oraz korelacje między zmiennymi. Wykresy graficzne i statystyki potwierdziły zależność między wysokością a głębokością drzewa.

Cały kod wraz z liniami

```
1. import random
2. import math
3. import matplotlib.pyplot as plt
4.
5.
6.
7. def losuj(lista):
8.     for i in range(len(lista) - 1, 0, -1):
9.         j = random.randint(0, i)
10.         lista[i], lista[j] = lista[j], lista[i]
11.
12. class Node:
13.     def __init__(self, v):
14.         self.value = v
15.         self.left = None
16.         self.right = None
17.
18. class BST:
19.     def __init__(self):
20.         self.root = None
21.
22.     def dodaj(self, v):
23.         if self.root is None:
24.             self.root = Node(v)
25.         else:
26.             self._dodaj(self.root, v)
27.
28.     def _dodaj(self, node, v):
29.         if v < node.value:
```

```
30.             if node.left is None:
31.                 node.left = Node(v)
32.             else:
33.                 self._dodaj(node.left, v)
34.         elif v > node.value:
35.             if node.right is None:
36.                 node.right = Node(v)
37.             else:
38.                 self._dodaj(node.right, v)
39.
40.     def wysokosc(self):
41.         def f(n):
42.             if n is None:
43.                 return -1
44.             return 1 + max(f(n.left), f(n.right))
45.         return f(self.root)
46.
47.     def liscie(self):
48.         def f(n):
49.             if n is None:
50.                 return 0
51.             if n.left is None and n.right is None:
52.                 return 1
53.             return f(n.left) + f(n.right)
54.         return f(self.root)
55.
56.     def srednia_glebokosc(self):
57.         def f(n, g):
58.             if n is None:
59.                 return (0, 0)
```

```
            l_sum, l_ile = f(n.left, g + 1)
            r_sum, r_ile = f(n.right, g + 1)
            return (l_sum + r_sum + g, l_ile + r_ile + 1)
        suma, ile = f(self.root, 0)
        return suma / ile if ile > 0 else 0

def std(dane):
    s = sum(dane) / len(dane)
    return math.sqrt(sum((x - s) ** 2 for x in dane) / len(dane))

def korelacja(x, y):
    n = len(x)
    sx = sum(x) / n
    sy = sum(y) / n
    licznik = sum((x[i] - sx)*(y[i] - sy) for i in range(n))
    mianownik = math.sqrt(sum((x[i] - sx)**2 for i in range(n)) * sum((y[i] - sy)**2 for i in range(n)))
    return licznik / mianownik if mianownik != 0 else 0

def iqr(dane):
    dane_sorted = sorted(dane)
    n = len(dane_sorted)
    q1_index = n // 4
    q3_index = (3 * n) // 4
    return dane_sorted[q3_index] - dane_sorted[q1_index]

if __name__ == "__main__":
    h, l, d = [], [], []
    for p in range(300):
        dane = list(range(10000))
        losuj(dane)
```

```
90.         drzewo = BST()
91.         for x in dane:
92.             drzewo.dodaj(x)
93.         h.append(drzewo.wysokosc())
94.         l.append(drzewo.liscie())
95.         d.append(drzewo.srednia_glebokosc())
96.
97.
98.
99.     with open("wyniki_bst.txt", "w") as f:
100.         for i in range(len(h)):
101.             f.write(f"Próba {i+1}: Wysokość = {h[i]}, Liście = {l[i]}, Średnia głębokość = {round(d[i], 3)}\n")
102.
103.
104.     print("\nKorelacje:")
105.     print("wysokosc vs liscie:", round(korelacja(h, l), 3))
106.     print("wysokosc vs glebokosc:", round(korelacja(h, d), 3))
107.     print("liscie vs glebokosc:", round(korelacja(l, d), 3))
108.
109.     print("\nOdchylenie standardowe wysokosci:", round(std(h), 3))
110.     print("Rozstęp międzykwartylowy wysokości:", iqr(h))
111.
112.
113.
114. def zapisz_wykresy(heights, leaves, depths):
115.
116.     plt.figure()
117.     plt.hist(heights, bins=20, color='skyblue', edgecolor='black')
118.     plt.title("Histogram wysokości drzewa")
119.     plt.xlabel("Wysokość")
```

```
120.     plt.ylabel("Liczba przypadków")
121.     plt.show()
122.
123.
124.     plt.figure()
125.     plt.hist(leaves, bins=20, color='lightgreen', edgecolor='black')
126.     plt.title("Histogram liczby liści")
127.     plt.xlabel("Liście")
128.     plt.ylabel("Liczba przypadków")
129.     plt.show()
130.
131.
132.     plt.figure()
133.     plt.hist(depths, bins=20, color='salmon', edgecolor='black')
134.     plt.title("Histogram średniej głębokości")
135.     plt.xlabel("Średnia głębokość")
136.     plt.ylabel("Liczba przypadków")
137.     plt.show()
138.
139.
140. zapisz_wykresy(h, l, d)
```