AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

Wyszukiwanie Geometryczne

QuadTree i KdTree

Dokumentacja

Weronika Wojtas, Radosław Rolka
Algorytmy geometryczne
Informatyka WI AGH, II rok

2023

Spis treści

I. Wstęp

1 Informacje wstępne

Celem tej dokumentacji jest przedstawienie zaawansowanych technik wyszukiwania geometrycznego, z naciskiem na implementację i zastosowanie KdTree oraz QuadTree oraz ich porównanie.

Projekt został zaimplementowany w języku Python 3.9 z wykorzystaniem aplikacji Jupyter Notebook. Kod źródłowy wraz z dokumentacją zamieszczony jest w repozytorium GitHub.

2 Wymagania techniczne

- python >= 3.9
- numpy >= 1.25.2
- pandas >= 2.0.3
- matplotlib >= 3.7.2
- notebook >= 6.5.4
- unittest (Python Standard Library)
- functools (Python Standard Library)
- math (Python Standard Library)
- copy (Python Standard Library)

3 Środowisko wykonawcze

	Komputer 1	Komputer 2
Procesor	Intel(R) Core(TM) I5-10300H 2.50GHz	Intel(R) Core(TM) I3-1115G4 3.00GHz
System operacyjny	Microsoft Windows 10 64bit ver 22H2	Microsoft Windows 11 Home 64bit ver 22H2

Tabela 1: Dane techniczne maszyn wykonawczych

Porównanie efektywności struktur zostało wykonane na Komputerze 1.

4 Zawarość programu

Program składa się z następujących plików:

- KdTree.py
- QuadTree.py
- TestManager.py
- Comparision.ipynb
- visualizer/
- utilities/
- tests/

- documentation/
- comparator/

II. Dokumentacja

W tym rozdziale znajduje się szczegółowa dokumentacja poszczególnych modułów programu, które są przewidziane do wykorzystania przez użytkownika. Klasy oraz metody, które nie są wymienione w tej dokumentacji, nie są przewidziane do bezpośredniego użycia.

1 Utilities

Ten moduł zapewnia podstawowe klasy elementów geometrycznych: punkt (Point) i prostokąt (Rectangle), które unifikują dane wejściowe do obu implementacji drzew oraz zapewniają walidację wprowadzonych danych.

1.1 Point

Reprezentacja niemutowalnego punktu o dodatniej liczbie wymiarów wraz z licznymi możliwościami wykorzystania i przetwarzania.

- **Point(point)**: Konstruktor punktu przyjmuje iterowalny obiekt jako argument i zapisuje jego wartości jako kolejne wymiary dla punktu.

  - point: iterowalny obiekt, którego wartości zostaną zapisane jako kolejne wymiary punktu.

  - RESULT: punkt zawierający wartości z point.

  Wywołuje ValueError("Point must have at least one dimension") jeśli obiekt jest pusty.

- **__eq__(self, other)**: Porównuje punkty na podstawie ich wymiarów, zwraca True jeśli wymiary są równe, w przeciwnym wypadku False. Umożliwia porównywanie punktów oraz innych iterowalnch struktur za pomocą operatorów == i !=.

  - other: punkt lub inny iterowalny obiekt, który zostanie porównany z punktem.

  - RESULT: True jeśli wymiary punktów są równe, False w przeciwnym wypadku.

- **__hash__(self)**: Zwraca hash punktu, umożliwiając jego użycie jako klucza w słownikach.

  - RESULT: hash punktu.

- **__str__(self) / __repr__(self)**: Zwraca string reprezentujący punkt w postaci (x1, x2, ..., xn), gdzie xi to kolejne wymiary punktu. Umożliwia wyświetlanie punktu za pomocą funkcji print().

  - RESULT: string reprezentujący punkt w postaci (x1, x2, ..., xn).

- **__len__(self)**: Zwraca liczbę wymiarów punktu.

  - RESULT: liczba wymiarów punktu.

- **__getitem__(self, key)**: Zwraca wymiar o indeksie key, umożliwiając dostęp do wymiarów punktu za pomocą operatora nawiasów kwadratowych.

  - key: indeks wymiaru.

  - RESULT: wymiar o indeksie key.

- **point(self)**: @property Zwraca listę zawierającą kopię wymiarów punktu.

- **x(self)** @property Zwraca pierwszy wymiar punktu.

- **y(self)**: @property Zwraca drugi wymiar punktu, jeśli istnieje.

- **follows(self, other)**: Porównuje punkty na podstawie ich wymiarów, zwraca True jeśli punkt jest mniejszy od other w każdym wymiarze, w przeciwnym wypadku False. Zezwala na porównywanie obiektów iterowalnych.

  - other: punkt lub inny iterowalny obiekt, który zostanie porównany z punktem.

  - RESULT: True jeśli punkt jest większy od other w każdym wymiarze, False w przeciwnym wypadku.

  Wywołuje ValueError("Can only compare Points of the same dimensionality") jeśli wymiary punktów nie są równe.

- **precedes(self, other)**: Porównuje punkty na podstawie ich wymiarów, zwraca True jeśli punkt jest większy od other w każdym wymiarze, w przeciwnym wypadku False. Zezwala na porównywanie obiektów iterowalnych.

  - other: punkt lub inny iterowalny obiekt, który zostanie porównany z punktem.

  - RESULT: True jeśli punkt jest większy od other w każdym wymiarze, False w przeciwnym wypadku.

  Wywołuje ValueError("Can only compare Points of the same dimensionality") jeśli wymiary punktów nie są równe.

- **distance(self, other)**: Oblicza odległość między punktami. Zezwala na porównywanie obiektów iterowalnych.

  - other: punkt lub inny iterowalny obiekt, od którego zostanie obliczona odległość.

  - RESULT: odległość między punktami.

  Wywołuje ValueError("Can only compare Points of the same dimensionality") jeśli wymiary punktów nie są równe.

- **minimum(self, other)**: Wyznacza punkt zawierający najmniejsze wartości z obu punktów. Zezwala na porównywanie obiektów iterowalnych.

  - other: punkt lub inny iterowalny obiekt, z którym zostanie wyznaczony punkt zawierający najmniejsze wartości.

  - RESULT: punkt zawierający najmniejsze wartości z obu punktów.

  Wywołuje ValueError("Can only compare Points of the same dimensionality") jeśli wymiary punktów nie są równe.

- **maximum(self, other)**: Wyznacza punkt zawierający największe wartości z obu punktów. Zezwala na porównywanie obiektów iterowalnych.

  - other: punkt lub inny iterowalny obiekt, z którym zostanie wyznaczony punkt zawierający największe wartości.

  - RESULT: punkt zawierający największe wartości z obu punktów.

  Wywołuje ValueError("Can only compare Points of the same dimensionality") jeśli wymiary punktów nie są równe.

1.2 Rectangle

Reprezentacja niemutowalnego prostokąta o dodatniej liczbie wymiarów, zapisanego jako dwa przeciwległe wierzchołki: lewy dolny (lowerleft) i prawy górny (upperright).

- **Rectangle(lowerleft, upperright)**: Konstruktor prostokąta przyjmuje dwa punkty (lub obiekty iterowalne) jako argumenty i zapisuje je jako przeciwległe wierzchołki prostokąta.

  - lowerleft: punkt (lub obiekt iterowalny), który zostanie zapisany jako lewy dolny wierzchołek prostokąta.

  - upperright: punkt (lub obiekt iterowalny), który zostanie zapisany jako prawy górny wierzchołek prostokąta.

  - RESULT: prostokąt zawierający punkty lowerleft i upperright.

Wywołuje ValueError("Points must have the same dimensionality") jeśli wymiary punktów nie są równe.

Wywołuje ValueError("LowerLeft point must precede the UpperRight point") jeśli któryś z wymiarów punktu lowerleft jest większy niż odpowiadający mu wymiar punktu upperright.

- **__eq__(self, other)**: Porównuje prostokąty na podstawie ich wierzchołków, zwraca True jeśli wierzchołki są równe, w przeciwnym wypadku False. Umożliwia porównywanie za pomocą operatorów == i !=.

  - other: dowolny obiekt, który zostanie porównany z prostokątem.

  - RESULT: True jeśli wierzchołki prostokątów są równe, False w przeciwnym wypadku.

- **__hash__(self)**: Zwraca hash prostokąta, umożliwiając jego użycie jako klucza w słownikach.

  - RESULT: hash prostokąta.

- **__str__(self) / __repr__(self)**: Zwraca string reprezentujący prostokąt w postaci [(x1, x2, ..., xn), (y1, y2, ..., yn)].

  - RESULT: string reprezentujący prostokąt w postaci [(x1, x2, ..., xn), (y1, y2, ..., yn)].

- **__len__(self)**: Zwraca liczbę wymiarów prostokąta.

  - RESULT: liczba wymiarów prostokąta.

- **lowerleft(self)**: @property Zwraca punkt lowerleft.

- **upperright(self)**: @property Zwraca punkt upperright.

- **from_points(cls, points)**: @classmethod Tworzy najmniejszy prostokąt zawierający wszystkie punkty (lub obiekty iterowalne) z listy points.

  - points: lista punktów (lub obiektów iterowalnych), które zostaną zawarte w prostokącie.

  - RESULT: najmniejszy prostokąt zawierający wszystkie punkty z listy points.

  Wywołuje ValueError("Cannot create a Rectangle from an empty list of points") jeśli lista points jest pusta.

  Wywołuje ValueError("Points must have the same dimensionality") jeśli wymiary punktów nie są równe.

- **does_intersect(self, other)**: Sprawdza czy dwa prostokąty przecinają się (mają conajmniej jeden punkt wspólny), zwraca True jeśli tak, w przeciwnym wypadku False.

  - other: prostokąt, który zostanie sprawdzony pod kątem przecięcia z prostokątem.

  - RESULT: True jeśli prostokąty przecinają się, False w przeciwnym wypadku.

  Wywołuje ValueError("Can only check intersection with another Rectangle") jeśli other nie jest prostokątem.

  Wywołuje ValueError("Can only check intersection with a Rectangle of the same dimensionality") jeśli wymiary prostokątów nie są równe.

- **contains(self, object)**: Sprawdza czy punkt lub prostokąt jest całkowicie zawarty w prostokącie, zwraca True jeśli tak, w przeciwnym wypadku False. Jeśli obiekt nie jest prostokątem, funkcja uznaje go za punkt.

  - object: punkt lub prostokąt, który zostanie sprawdzony pod kątem zawierania w prostokącie.

  - RESULT: True jeśli obiekt jest zawarty w prostokącie, False w przeciwnym wypadku.

- **divide(self, dimension, value)**: Dzieli prostokąt na dwa prostokąty wzdłuż wymiaru dimension i wartości value, zwraca krotkę dwóch prostokątów.

  - dimension: wymiar, wzdłuż którego następuje podział prostokąta.

  - value: wartość, względem której następuje podział prostokąta.

  - RESULT: krotka dwóch prostokątów.

Wywołuje ValueError("Dimension must be between 0 and len(self)-1") jeśli wymiar dimension jest mniejszy niż 0 lub większy niż liczba wymiarów prostokąta - 1.

Wywołuje ValueError("Value must be between self.lowerleft[dimension] and self.upperright[dimension]") jeśli wartość value jest mniejsza niż wartość wymiaru dimension punktu lowerleft lub większa niż wartość wymiaru dimension punktu upperright.

- **intersection(self, other)**: Zwraca prostokąt będący częścią wspólną dwóch prostokątów lub None jeśli prostokąty się nie przecinają.

  - other: prostokąt, z którym zostanie wyznaczona część wspólna.

  - RESULT: prostokąt będący częścią wspólną dwóch prostokątów lub None jeśli prostokąty się nie przecinają.

  Wywołuje ValueError("Can only compute intersection with another Rectangle") jeśli other nie jest prostokątem.

  Wywołuje ValueError("Can only compute intersection with a Rectangle of the same dimensionality") jeśli wymiary prostokątów nie są równe.

- **vertices2D(self)**: @property Zwraca listę punktów będących wierzchołkami prostokąta, jeśli prostokąt jest dwuwymiarowy.

  Wywołuje ValueError("Can only compute vertices of a 2D rectangle") jeśli prostokąt nie jest dwuwymiarowy.

- **opposite(self, point, depth)**: Zwraca dwa punkty leżące na przeciwległych bokach prostokąta o dwóch wymiarach, wzdłuż wymiaru depth, takie że point również leży na tym odcinku.

  - point: punkt, względem którego zostaną wyznaczone punkty leżące na przeciwległych bokach prostokąta.

  - depth: wymiar, wzdłuż którego zostaną wyznaczone punkty leżące na przeciwległych bokach prostokąta.

  Wywołuje ValueError("Can only compute opposite points of a 2D rectangle") jeśli prostokąt nie jest dwuwymiarowy.

- center(self): Zwraca punkt będący środkiem prostokąta.

  - RESULT: punkt będący środkiem prostokąta.

- **to_quaters(self)**: Dzieli prostokąt na cztery równe podprostokąty, zwraca listę podprostokątów klasy Rectangle.

  - RESULT: lista podprostokątów klasy Rectangle.

  Wywołuje ValueError("Can only divide a 2D rectangle") jeśli prostokąt nie jest dwuwymiarowy.

2 QuadTree

Plik zawiera implementację drzewa QuadTree (QuadTree) oraz jego bliźniaczą wersję (QuadTree_visualizer), która umożliwia wizualizację drzewa w postaci gifów.

2.1 QuadTree

Klasa implementująca drzewo QuadTree wraz z metodami wyszukiwania oraz weryfikacją danych wejściowych.

- **QuadTree(points, max_capacity=1, points_in_node=False)**: Konstruktor drzewa przyjmuje listę punktów (obiekty klasy Point lub obiektów iterowalnych) jako argument i tworzy drzewo zawierające te punkty.

  - points: lista punktów (obiektów klasy Point lub obiektów iterowalnych), które będą przechowywane w drzewie.

  - max_capacity: maksymalna liczba punktów w węźle, po przekroczeniu której następuje podział węzła.

- points_in_node: określa czy w każdym węźle przechowywana jest lista znajdujących się w nim punktów (True), czy tylko w liściach (False).

- RESULT: drzewo QuadTree zawierające punkty z listy points.

Wywołuje ValueError("The list of points is empty.") jeśli lista points jest pusta.

Wywołuje ValueError("The points have dimensions different than 2.") jeśli wymiary punktów nie są równe 2.

Wywołuje ValueError("The max capacity must be greater than 0.") jeśli max_capacity jest mniejsze niż 1.

Wywołuje ValueError("Not all points are unique.") jeśli lista points zawiera duplikaty.

- **if_contains(self, point)**: Sprawdza czy drzewo zawiera punkt (obiekt klasy Point lub obiekt iterowalny), zwraca True jeśli tak, w przeciwnym wypadku False.

  - point: punkt (obiekt klasy Point lub obiekt iterowalny), który ma zostać wyszukany w drzewie.

  - RESULT: True jeśli drzewo zawiera punkt, False w przeciwnym wypadku.

  Wywołuje ValueError("The point has different dimension than 2.") jeśli wymiary punktu nie są równe 2.

- **search_in_rectangle(self, rectangle, raw=False)**: Wyszukuje punkty znajdujące się w prostokącie (obiekt klasy Rectangle), zwraca listę punktów (obiektów klasy Point) lub listę list punktów (obiektów klasy Point) w zależności od wartości parametru raw.

  - rectangle: prostokąt (obiekt klasy Rectangle), w którym wyszukiwane są punkty.

  - raw: określa czy zwracana lista ma zawierać obiekty klasy Point (False) lub listy współrzędnych punktów (True).

  - RESULT: lista punktów (obiektów klasy Point) lub lista list punktów (obiektów klasy Point) w zależności od wartości parametru raw.

  Wywołuje ValueError("The rectanangle is not a Rectangle object.") jeśli rectangle nie jest obiektem klasy Rectangle.

  Wywołuje ValueError("The rectangle has different dimension than 2.") jeśli wymiary prostokąta nie są równe 2.

2.2 QuadTree_visualizer

Klasa implementująca drzewo QuadTree wraz z metodami wyszukiwania oraz weryfikacją danych wejściowych. Dodatkowo umożliwia wizualizację drzewa w postaci gifów.

- **QuadTree_visualizer(points, max_capacity=1, points_in_node=False, visualize_gif=True, title="QuadTree", filename="QuadTree-construction")**: Konstruktor drzewa przyjmuje listę punktów (obiekty klasy Point lub obiektów iterowalnych) jako argument i tworzy drzewo zawierające te punkty. Dodatkowo tworzy obiekt klasy Visualizer i zapisuje plik o rozszerzeniu .gif w tym samym folderze.

  - points: lista punktów (obiektów klasy Point lub obiektów iterowalnych), które będą przechowywane w drzewie.

  - max_capacity: maksymalna liczba punktów w węźle, po przekroczeniu której następuje podział węzła.

  - points_in_node: określa czy w każdym węźle przechowywana jest lista znajdujących się w nim punktów (True), czy tylko w liściach (False).

  - visualize_gif: określa czy wizualizacja drzewa ma zostać zapisana w pliku .gif, czy tylko wyświetlona.

  - title: tytuł wyświetlany na wykresie.

  - filename: nazwa pliku .gif, w którym zapisywana jest wizualizacja.

  - RESULT: drzewo QuadTree zawierające punkty z listy points.

  Wywołuje ValueError("The list of points is empty.") jeśli lista points jest pusta.

Wywołuje ValueError("The points have dimensions different than 2.") jeśli wymiary punktów nie są równe 2.

Wywołuje ValueError("The max capacity must be greater than 0.") jeśli max_capacity jest mniejsze niż 1.

Wywołuje ValueError("Not all points are unique.") jeśli lista points zawiera duplikaty.

Legenda:

- **Granatowe odcinki**: odcinki reprezentujące podział węzła.
- **Niebieskie punkty**: punkty nieprzetworzone.
- **Czerwone punkty**: punkty aktualnie przetwarzane.
- **Pomarańczowe punkty**: punkty przetworzone.
- **Żółty obszar**: obszar aktualnie przetwarzany.

- **if_contains(self, point, visualize_gif=True, title="QuadTree", filename="QuadTree-contains")**: Sprawdza czy drzewo zawiera punkt (obiekt klasy Point lub obiekt iterowalny), zwraca True jeśli tak, w przeciwnym wypadku False. Dodatkowo tworzy obiekt klasy Visualizer i zapisuje plik o rozszerzeniu .gif w tym samym folderze.

  - point: punkt (obiekt klasy Point lub obiekt iterowalny), który ma zostać wyszukany w drzewie.

  - visualize_gif: określa czy wizualizacja drzewa ma zostać zapisana w pliku .gif, czy tylko wyświetlona.

  - title: tytuł wyświetlany na wykresie.

  - filename: nazwa pliku .gif, w którym zapisywana jest wizualizacja.

  - RESULT: True jeśli drzewo zawiera punkt, False w przeciwnym wypadku.

  Wywołuje ValueError("The point has different dimension than 2.") jeśli wymiary punktu nie są równe 2.

  Legenda:

  - **Granatowe odcinki**: odcinki reprezentujące podział węzła.
  - **Niebieskie punkty**: punkty nieprzetworzone.
  - **Czerwone punkty**: punkty aktualnie przetwarzane.
  - **Pomarańczowe punkty**: punkty przetworzone.
  - **Zielony punkt**: punkt, który został znaleziony.
  - **Czarny punkt**: lokalizacja punktu, który nie został znaleziony.
  - **Żółty obszar**: obszar aktualnie przetwarzany.

- **search_in_rectangle(self, rectangle, visualize_gif=True, raw=False, title="QuadTree", filename= "QuadTree-search")**: Wyszukuje punkty znajdujące się w prostokącie (obiekt klasy Rectangle), zwraca listę punktów (obiektów klasy Point) lub listę list punktów (obiektów klasy Point) w zależności od wartości parametru raw. Dodatkowo tworzy obiekt klasy Visualizer i zapisuje plik o rozszerzeniu .gif w tym samym folderze.

  - rectangle: prostokąt (obiekt klasy Rectangle), w którym wyszukiwane są punkty.

  - visualize_gif: określa czy wizualizacja drzewa ma zostać zapisana w pliku .gif, czy tylko wyświetlona.

  - raw: określa czy zwracana lista ma zawierać obiekty klasy Point (False) lub listy współrzędnych punktów (True).

  - title: tytuł wyświetlany na wykresie.

  - filename: nazwa pliku .gif, w którym zapisywana jest wizualizacja.

  - RESULT: lista punktów (obiektów klasy Point) lub lista list punktów (obiektów klasy Point) w zależności od wartości parametru raw.

Wywołuje ValueError("The rectanangle is not a Rectangle object.") jeśli rectangle nie jest obiektem klasy Rectangle.

Wywołuje ValueError("The rectangle has different dimension than 2.") jeśli wymiary prostokąta nie są równe 2.

Legenda:

- **Granatowe odcinki**: odcinki reprezentujące podział węzła.
- **Czerwone odcinki**: odcinki reprezentujące obszar przeszukiwania.
- **Niebieskie punkty**: punkty nieprzetworzone.
- **Czerwone punkty**: punkty aktualnie przetwarzane.
- **Pomarańczowe punkty**: punkty przetworzone.
- **Zielony punkt**: punkt, który został znaleziony.
- Żółty obszar: obszar aktualnie przetwarzany.

3 KdTree

Plik zawiera implementację drzewa KdTree (KdTree) oraz jego bliźniaczą wersję (KdTree_visualizer), która umożliwia wizualizację drzewa w postaci gifów.

3.1 KdTree

Klasa implementująca drzewo KdTree wraz z metodami wyszukiwania oraz weryfikacją danych wejściowych.

- **KdTree(points, depth=0, points_in_node=False)**: Konstruktor drzewa przyjmuje listę punktów (obiekty klasy Point lub obiektów iterowalnych) jako argument i tworzy drzewo zawierające te punkty.

  - points: lista punktów (obiektów klasy Point lub obiektów iterowalnych), które będą przechowywane w drzewie.

  - depth: głębokość węzła, operacja $\%len(point)$ wskazuje na wymiar, w którym następuje podział węzła.

  - points_in_node: określa czy w każdym węźle przechowywana jest lista znajdujących się w nim punktów (True), czy tylko w liściach (False).

  - RESULT: drzewo KdTree zawierające punkty z listy points.

  Wywołuje ValueError("The list of points is empty.") jeśli lista points jest pusta.

  Wywołuje ValueError("The points have different dimensions.") jeśli wymiary punktów nie są równe.

- **if_contains(self, point)**: Sprawdza czy drzewo zawiera punkt (obiekt klasy Point lub obiekt iterowalny), zwraca True jeśli tak, w przeciwnym wypadku False.

  - point: punkt (obiekt klasy Point lub obiekt iterowalny), który ma zostać wyszukany w drzewie.

  - RESULT: True jeśli drzewo zawiera punkt, False w przeciwnym wypadku.

  Wywołuje ValueError("The point has different dimension than the points in the tree.") jeśli wymiary punktu nie są równe wymiarom punktów w drzewie.

- **search_in_rectangle(self, rectangle, raw=False)**: Wyszukuje punkty znajdujące się w prostokącie (obiekt klasy Rectangle), zwraca listę punktów (obiektów klasy Point) lub listę list punktów (obiektów klasy Point) w zależności od wartości parametru raw.

  - rectangle: prostokąt (obiekt klasy Rectangle), w którym wyszukiwane są punkty.

  - raw: określa czy zwracana lista ma zawierać obiekty klasy Point (False) lub listy współrzędnych punktów (True).

  - RESULT: lista punktów (obiektów klasy Point) lub lista list punktów (obiektów klasy Point) w zależności od wartości parametru raw.

Wywołuje ValueError("The rectanangle is not a Rectangle object.") jeśli rectangle nie jest obiektem klasy Rectangle.

Wywołuje ValueError("The rectangle has different dimension than the points in the tree.") jeśli wymiary prostokąta nie są równe wymiarom punktów w drzewie.

3.2 KdTree_visualizer

Klasa implementująca drzewo KdTree wraz z metodami wyszukiwania oraz weryfikacją danych wejściowych. Dodatkowo umożliwia wizualizację drzewa w postaci gifów.

- **KdTree_visualizer(points, depth=0, points_in_node=False, visualize_gif=True, title="KdTree", filename="KdTree-construction")**: Konstruktor drzewa przyjmuje listę punktów (obiekty klasy Point lub obiektów iterowalnych) jako argument i tworzy drzewo zawierające te punkty. Dodatkowo tworzy obiekt klasy Visualizer i zapisuje plik o rozszerzeniu .gif w tym samym folderze.

  - points: lista punktów (obiektów klasy Point lub obiektów iterowalnych), które będą przechowywane w drzewie.

  - depth: głębokość węzła, operacja $\%len(point)$ wskazuje na wymiar, w którym następuje podział węzła.

  - points_in_node: określa czy w każdym węźle przechowywana jest lista znajdujących się w nim punktów (True), czy tylko w liściach (False).

  - visualize_gif: określa czy wizualizacja drzewa ma zostać zapisana w pliku .gif (True), czy tylko wyświetlona (False).

  - title: tytuł wyświetlany na wykresie.

  - filename: nazwa pliku .gif, w którym zapisywana jest wizualizacja.

  - RESULT: drzewo KdTree zawierające punkty z listy points.

  Wywołuje ValueError("The list of points is empty.") jeśli lista points jest pusta.

  Wywołuje ValueError("The points have different dimensions.") jeśli wymiary punktów nie są równe.

  Wywołuje ValueError("Not all points are unique.") jeśli lista points zawiera duplikaty.

  Legenda:

  - **Granatowe odcinki**: odcinki reprezentujące podział węzła.
  - **Niebieskie punkty**: punkty nieprzetworzone.
  - **Czerwone punkty**: punkty aktualnie przetwarzane.
  - **Pomarańczowe punkty**: punkty przetworzone.
  - **Żółty obszar**: obszar aktualnie przetwarzany.

- **if_contains(self, point, visualize_gif=True, title="KdTree", filename="KdTree-contains")**: Sprawdza czy drzewo zawiera punkt (obiekt klasy Point lub obiekt iterowalny), zwraca True jeśli tak, w przeciwnym wypadku False. Dodatkowo tworzy obiekt klasy Visualizer i zapisuje plik o rozszerzeniu .gif w tym samym folderze.

  - point: punkt (obiekt klasy Point lub obiekt iterowalny), który ma zostać wyszukany w drzewie.

  - visualize_gif: określa czy wizualizacja drzewa ma zostać zapisana w pliku .gif (True), czy tylko wyświetlona (False).

  - title: tytuł wyświetlany na wykresie.

  - filename: nazwa pliku .gif, w którym zapisywana jest wizualizacja.

  - RESULT: True jeśli drzewo zawiera punkt, False w przeciwnym wypadku.

  Wywołuje ValueError("The point has different dimension than the points in the tree.") jeśli wymiary punktu nie są równe wymiarom punktów w drzewie.

  Legenda:

  - **Granatowe odcinki**: odcinki reprezentujące podział węzła.
  - **Niebieskie punkty**: punkty nieprzetworzone.
  - **Czerwone punkty**: punkty aktualnie przetwarzane.
  - **Pomarańczowe punkty**: punkty przetworzone.
  - **Zielony punkt**: punkt, który został znaleziony.
  - **Czarny punkt**: lokalizacja punktu, który nie został znaleziony.
  - **Żółty obszar**: obszar aktualnie przetwarzany.

- **search_in_rectangle(self, rectangle, raw=False, visualize_gif=True, title="KdTree", filename= "KdTree-search")**: Wyszukuje punkty znajdujące się w prostokącie (obiekt klasy Rectangle), zwraca listę punktów (obiektów klasy Point) lub listę list punktów (obiektów klasy Point) w zależności od wartości parametru raw. Dodatkowo tworzy obiekt klasy Visualizer i zapisuje plik o rozszerzeniu .gif w tym samym folderze.

  - rectangle: prostokąt (obiekt klasy Rectangle), w którym wyszukiwane są punkty.

  - raw: określa czy zwracana lista ma zawierać obiekty klasy Point (False) lub listy współrzędnych punktów (True).

  - visualize_gif: określa czy wizualizacja drzewa ma zostać zapisana w pliku .gif (True), czy tylko wyświetlona (False).

  - title: tytuł wyświetlany na wykresie.

  - filename: nazwa pliku .gif, w którym zapisywana jest wizualizacja.

  - RESULT: lista punktów (obiektów klasy Point) lub lista list punktów (obiektów klasy Point) w zależności od wartości parametru raw.

  Wywołuje ValueError("The rectanangle is not a Rectangle object.") jeśli rectangle nie jest obiektem klasy Rectangle.

  Wywołuje ValueError("The rectangle has different dimension than the points in the tree.") jeśli wymiary prostokąta nie są równe wymiarom punktów w drzewie.

  Legenda:

  - **Granatowe odcinki**: odcinki reprezentujące podział węzła.
  - **Czerwone odcinki**: odcinki reprezentujące obszar przeszukiwania.
  - **Niebieskie punkty**: punkty nieprzetworzone.
  - **Czerwone punkty**: punkty aktualnie przetwarzane.
  - **Pomarańczowe punkty**: punkty przetworzone.
  - **Zielony punkt**: punkt, który został znaleziony.
  - **Żółty obszar**: obszar aktualnie przetwarzany.

4 Comparator

Moduł zawiera implementację porównania drzew QuadTree i KdTree oraz klasę generującą przypadki testowe.

4.1 CaseGenerator

Klasa generująca przypadki testowe dla struktur QuadTree i KdTree.

- **CaseGenerator()**: Konstruktor klasy CaseGenerator.

- **uniform_distribution(self, quantity, rectangle, raw=True)**: Generuje punkty z rozkładem jednorodnym wewnątrz prostokąta.

  - quantity: liczba punktów do wygenerowania.

  - rectangle: prostokąt (obiekt klasy Rectangle), w którym mają zostać wygenerowane punkty.

  - raw: określa czy zwracana lista ma zawierać obiekty klasy Point (False) lub listy współrzędnych punktów (True).

  - RESULT: lista punktów (obiektów klasy Point) lub lista list punktów (obiektów klasy Point) w zależności od wartości parametru raw.

  Wywołuje ValueError("quantity must be positive") jeśli quantity jest mniejsze od 0.

  Wywołuje ValueError("The rectanangle is not a Rectangle object.") jeśli rectangle nie jest obiektem klasy Rectangle.

- **normal_distribution(self, quantity, rectangle, raw=True, mu=None, sigma=None)**:

  Generuje punkty z rozkładem normalnym wewnątrz prostokąta.

  - quantity: liczba punktów do wygenerowania.

  - rectangle: prostokąt (obiekt klasy Rectangle), w którym mają zostać wygenerowane punkty.

  - raw: określa czy zwracana lista ma zawierać obiekty klasy Point (False) lub listy współrzędnych punktów (True).

  - mu: wartość oczekiwana rozkładu normalnego, w przypadku None wartość oczekiwana jest wyznaczana jako środek prostokąta. Musi być listą o takiej samej długości jak wymiary prostokąta.

  - sigma: odchylenie standardowe rozkładu normalnego, w przypadku None odchylenie standardowe jest wyznaczane jako 1/6 długości boku prostokąta. Przy dużych wartościach sigma punkty mogą znajdować się poza prostokątem. Musi być listą o takiej samej długości jak wymiary prostokąta.

  - RESULT: lista punktów (obiektów klasy Point) lub lista list punktów (obiektów klasy Point) w zależności od wartości parametru raw.

  Wywołuje ValueError("quantity must be positive") jeśli quantity jest mniejsze od 0.

  Wywołuje ValueError("The rectanangle is not a Rectangle object.") jeśli rectangle nie jest obiektem klasy Rectangle.

  Wywołuje ValueError("mu must be list") jeśli mu nie jest listą.

  Wywołuje ValueError("mu must have the same dimensionality as rectangle") jeśli wymiary mu nie są równe wymiarom prostokąta.

  Wywołuje ValueError("sigma must be list") jeśli sigma nie jest listą.

  Wywołuje ValueError("sigma must have the same dimensionality as rectangle") jeśli wymiary sigma nie są równe wymiarom prostokąta.

- **grid_distribution(self, quantity, rectangle, raw=True)**: Generuje równo rozłożone punkty w siatce wewnątrz prostokąta.

  - quantity: krotka (columns, rows) określająca liczbę kolumn i wierszy punktów do wygenerowania.

  - rectangle: prostokąt (obiekt klasy Rectangle), w którym mają zostać wygenerowane punkty.

  - raw: określa czy zwracana lista ma zawierać obiekty klasy Point (False) lub listy współrzędnych punktów (True).

  - RESULT: lista punktów (obiektów klasy Point) lub lista list punktów (obiektów klasy Point) w zależności od wartości parametru raw.

  Wywołuje ValueError("Quantity must be a tuple of two positive integers") jeśli quantity nie jest krotką dwóch liczb całkowitych większych od 0.

  Wywołuje ValueError("columns must be positive") jeśli columns jest mniejsze od 0.

Wywołuje ValueError("rows must be positive") jeśli rows jest mniejsze od 0.

Wywołuje ValueError("The rectanangle is not a Rectangle object.") jeśli rectangle nie jest obiektem klasy Rectangle.

- **cluster_distribution(self, quantity, clusters, raw=True)**: Generuje punkty w klastrach wewnątrz prostokąta.

  - quantity: liczba punktów do wygenerowania w każdym klastrze.

  - clusters: lista prostokątów (obiektów klasy Rectangle), w których mają zostać wygenerowane punkty.

  - raw: określa czy zwracana lista ma zawierać obiekty klasy Point (False) lub listy współrzędnych punktów (True).

  - RESULT: lista punktów (obiektów klasy Point) lub lista list punktów (obiektów klasy Point) w zależności od wartości parametru raw.

  Wywołuje ValueError("quantity must be positive") jeśli quantity jest mniejsze od 0.

  Wywołuje ValueError("The rectanangle is not a Rectangle object.") jeśli element listy clusters nie jest obiektem klasy Rectangle.

- **outliers_distribution(self, quantities, rectangle, raw=True)**: Generuje punkty wewnątrz prostokąta wraz z punktami odstającymi. Punkty są generowane w środku dwukrotnie mniejszego prostokąta przy zachowaniu tego samego punktu centralnego, a punkty odstające na przestrzeni całego prostokąta.

  - quantities: krotka (quantity, outliers) określająca liczbę punktów i punktów odstających do wygenerowania.

  - outliers: liczba punktów odstających do wygenerowania.

  - rectangle: prostokąt (obiekt klasy Rectangle), w którym mają zostać wygenerowane punkty.

  - raw: określa czy zwracana lista ma zawierać obiekty klasy Point (False) lub listy współrzędnych punktów (True).

  - RESULT: lista punktów (obiektów klasy Point) lub lista list punktów (obiektów klasy Point) w zależności od wartości parametru raw.

  Wywołuje ValueError("quantity must be positive") jeśli quantity jest mniejsze od 0.

  Wywołuje ValueError("outliers must be positive") jeśli outliers jest mniejsze od 0.

  Wywołuje ValueError("The rectanangle is not a Rectangle object.") jeśli rectangle nie jest obiektem klasy Rectangle.

- **cross_distribution(self, quantity, rectangle, raw=True)**: Generuje punkty wewnątrz prostokąta na jego osiach symetrii względem jego boków.

  - quantity: krotka (vertical, horizontal) określająca liczbę punktów do wygenerowania w pionie i poziomie.

  - rectangle: prostokąt (obiekt klasy Rectangle), w którym mają zostać wygenerowane punkty.

  - raw: określa czy zwracana lista ma zawierać obiekty klasy Point (False) lub listy współrzędnych punktów (True).

  - RESULT: lista punktów (obiektów klasy Point) lub lista list punktów (obiektów klasy Point) w zależności od wartości parametru raw.

  Wywołuje ValueError("vertical must be positive") jeśli vertical jest mniejsze od 0.

  Wywołuje ValueError("horizontal must be positive") jeśli horizontal jest mniejsze od 0.

  Wywołuje ValueError("The rectanangle is not a Rectangle object.") jeśli rectangle nie jest obiektem klasy Rectangle.

- **rectangle_distribution(self, quantity, rectangle, raw=True)**: Generuje punkty na bokach prostokąta.

  - quantity: liczba punktów do wygenerowania.

- rectangle: prostokąt (obiekt klasy Rectangle), w którego bokach mają zostać wygenerowane punkty.

- raw: określa czy zwracana lista ma zawierać obiekty klasy Point (False) lub listy współrzędnych punktów (True).

- RESULT: lista punktów (obiektów klasy Point) lub lista list punktów (obiektów klasy Point) w zależności od wartości parametru raw.

Wywołuje ValueError("quantity must be positive") jeśli quantity jest mniejsze od 0.

Wywołuje ValueError("The rectanangle is not a Rectangle object.") jeśli rectangle nie jest obiektem klasy Rectangle.

4.2 Comparision.ipynb

Jupyter Notebook zawierający porównanie wydajności struktur QuadTree i KdTree wraz z generacją wykresów. Składa się z czterech części:

- **Sprawdzenie poprawności działania porównywanych struktur**: Sprawdzenie poprawności działania struktur QuadTree i KdTree za pomocą testów jednostkowych z modułu Tests.
- **Pomiary wydajności struktur QuadTree i KdTree dla różnych danych wejściowych oraz ich rozmiaru**: Pomiar czasu działania struktur QuadTree i KdTree dla różnych danych wejściowych wygenerowanych za pomocą CaseGenerator.
- **Generowanie wykresów**: Generowanie wykresów na podstawie danych z poprzedniej części.
- **Pomiary dla indywidualnych przypadków pod zastosowania drzewa**: Wykorzystanie drzew pod względem ich unikalnych możliwości.
- **Wizualizacja powyższych zbiorów, oraz siatki**: Wizualizacja zbiorów danych wejściowych oraz siatek utworzonych przez drzewa.

5 Tests

Moduł zawiera testy jednostkowe oraz integracyjne dla struktur QuadTree, KdTree oraz jednostkowe wszystkich komponentów.

5.1 TestManager

Klasa zawierająca testy integracyjne dla struktur QuadTree i KdTree.

- **TestManager(tree)**: Konstruktor klasy TestManager.

  - tree: drzewo (obiekt klasy QuadTree lub KdTree), które ma zostać przetestowane.

- **all_tests(self)**: Wykonuje wszystkie integracyjne zawarte w klasie TestManager. Przebieg jest wyświetlany w konsoli.

  - RESULT: True jeśli wszystkie testy zakończyły się sukcesem, False w przeciwnym wypadku.

- **all_unittests(self)**: Wykonuje wszystkie jednostkowe zawarte w klasie TestManager. Przebieg jest wyświetlany w konsoli. wykorzystuje bibliotekę unittest.

- **contain_point_int(self)**: Testuje metodę if_contains dla punktów całkowitoliczbowych. Przebieg jest wyświetlany w konsoli.

  - RESULT: krotka (good, all), gdzie good to liczba testów zakończonych sukcesem, a all to liczba wszystkich testów.

- **contain_point_float(self)**: Testuje metodę if_contains dla punktów zmiennoprzecinkowych. Przebieg jest wyświetlany w konsoli.

  - RESULT: krotka (good, all), gdzie good to liczba testów zakończonych sukcesem, a all to liczba wszystkich testów.

- **search_in_rectangle_int(self)**: Testuje metodę search_in_rectangle dla punktów całkowitoliczbowych. Przebieg jest wyświetlany w konsoli.

  - RESULT: krotka (good, all), gdzie good to liczba testów zakończonych sukcesem, a all to liczba wszystkich testów.

- **search_in_rectangle_float(self)**: Testuje metodę search_in_rectangle dla punktów zmiennoprzecinkowych. Przebieg jest wyświetlany w konsoli.

  - RESULT: krotka (good, all), gdzie good to liczba testów zakończonych sukcesem, a all to liczba wszystkich testów.

6 Visualizer

Moduł zawiera narzędzie do wizualizacji oraz przykładowe wizualizacje. Narzędzie pochodzi z repozytorium dostarczonego przez KN AGH BIT, link do repozytorium znajduje się w bibliografii.

6.1 main.py

Plik zawiera implementację klasy Visualizer, która umożliwia wizualizacje. Został oparty na możliwościach biblioteki matplotlib.

- **Visualizer()**: Konstruktor klasy Visualizer.
- **add_title(self, title)**: Dodaje tytuł wykresu.
  - title: tytuł wykresu.
- **add_grid(self)**: Dodaje siatkę do wykresu.
- **add_point(self, data, **kwargs)**: Dodaje punkt do wykresu.
  - data: punkt (obiekt iterowalny) który ma zostać dodany do wykresu.
  - kwargs: dodatkowe argumenty, które zostaną przekazane do funkcji matplotlib.pyplot.plot.
  - RESULT: obiekt klasy figure, który został dodany do wykresu.
- **add_line_segment(self, data, **kwargs)**: Dodaje odcinek do wykresu.
  - data: odcinek który ma zostać dodany do wykresu.
  - kwargs: dodatkowe argumenty, które zostaną przekazane do funkcji matplotlib.pyplot.plot.
  - RESULT: obiekt klasy figure, który został dodany do wykresu.
- **add_circle(self, data, **kwargs)**: Dodaje okrąg do wykresu.
  - data: okrąg który ma zostać dodany do wykresu.
  - kwargs: dodatkowe argumenty, które zostaną przekazane do funkcji matplotlib.pyplot.plot.
  - RESULT: obiekt klasy figure, który został dodany do wykresu.
- **add_polygon(self, data, **kwargs)**: Dodaje wielokąt do wykresu.
  - data: wielokąt który ma zostać dodany do wykresu.
  - kwargs: dodatkowe argumenty, które zostaną przekazane do funkcji matplotlib.pyplot.plot.
  - RESULT: obiekt klasy figure, który został dodany do wykresu.

- **add_line(self, data, **kwargs)**: Dodaje prostą do wykresu.
  - data: prosta która ma zostać dodana do wykresu.
  - kwargs: dodatkowe argumenty, które zostaną przekazane do funkcji matplotlib.pyplot.plot.
  - RESULT: obiekt klasy figure, który został dodany do wykresu.
- **add_half_line(self, data, **kwargs)**: Dodaje półprostą do wykresu.
  - data: półprosta która ma zostać dodana do wykresu.
  - kwargs: dodatkowe argumenty, które zostaną przekazane do funkcji matplotlib.pyplot.plot.
  - RESULT: obiekt klasy figure, który został dodany do wykresu.
- **remove_figure(self, figure)**: Usuwa figurę z wykresu.
  - figure: figura (obiekt klasy figure), która ma zostać usunięta z wykresu.
- **clear(self)**: Usuwa wszystkie figury z wykresu.
- **show(self)**: Wyświetla wykres.
- **save(self, filename='plot')**: Zapisuje wykres do pliku.
  - filename: nazwa pliku, w którym ma zostać zapisany wykres.
- **show_gif(self, interval=256)**: Wyświetla animację wykresu.
  - interval: czas trwania jednej klatki animacji w milisekundach.
  - RESULT: obiekt klasy Image, który został wyświetlony.
- **save_gif(self, filename='animation', interval=256)**: Zapisuje animację wykresu do pliku.
  - filename: nazwa pliku, w którym ma zostać zapisana animacja.
  - interval: czas trwania jednej klatki animacji w milisekundach.
  - RESULT: obiekt klasy Image, który został zapisany.

6.2 demo.ipynb

Plik Jupyter Notebook, który zawiera przykładowe wykorzystanie klasy Visualizer.

III. Instrukcja korzystania z programu

1 Utilities

1.1 Point

Aby utworzyć punkt należy utworzyć obiekt klasy Point, który przyjmuje listę współrzędnych jako argument.

```
from utilities.point import Point

point1 = Point([1, -2])
point2 = Point([3.4, -4, 5])
# ValueError: The point must have at least one dimension.
point3 = Point([]) # ValueError
```

Kod źródłowy 1: Konstruktor klasy Point.

Na obiekcie klasy Point można wykonywać różne operacje. Gdy do wykonania operacji wymagany jest inny punkt, to musi mieć on taki sam wymiar jak punkt na którym wykonywana jest operacja.

```
point1 = Point([1, -2])
point2 = Point([3.4, 5])
point3 = Point([3.4, -4, 5])

len(point1)             # 2
point3[2]               # 5
point2.y                # 5
point1.precedes(point2) # True
# ValueErorr: The points have different dimensions.
point1.maximum(point3)  # ValueError
```

Kod źródłowy 2: Przykładowe operacje na obiekcie klasy Point.

1.2 Rectangle

Aby utworzyć prostokąt należy utworzyć obiekt klasy Rectangle, który przyjmuje dwa punkty (obiekty klasy Point lub obiekty iterowalne) jako argumenty.

```
from utilities.rectangle import Rectangle

rectangle1 = Rectangle([1, -2], [7.4, 5])
rectangle2 = Rectangle(Point([1, -2]), Point([3.2, 5]))
rectangle3 = Rectangle([1, -2], Point([3.4, 6]))
# ValueError: The points have different dimensions.
rectangle4 = Rectangle([1, -2], [3.4, 6, 7]) # ValueError
# ValueError: LowerLeft point must precede the UpperRight point.
rectangle5 = Rectangle([3.4, 6], [1, -2])    # ValueError
```

Kod źródłowy 3: Konstruktor klasy Rectangle.

Przykładowe operacje z wykorzystaniem obiektu klasy Rectangle.

```
rectangle1 = Rectangle([1, -2], [7.4, 5])
rectangle2 = Rectangle(Point([1, -2.3]), Point([3.2, 5]))

rectangle1.lowerleft # Point([1, -2])
rectangle1.upperright # Point([7.4, 5])
Rectangle.from_points([2, 0.3], [9, 5]) # Rectangle([2, 0.3], [9, 5])
rectangle1.divide(0, 3) # [Rectangle([1, -2], [3, 5]), Rectangle([3, -2], [7.4, 5])]
rectanangle2.vertices # [Point([1, -2.3]), Point([3.2, -2.3]),
```

```
#  Point([3.2, 5]), Point([1, 5])]
```

Kod źródłowy 4: Operacje na obiekcie klasy Rectangle.

2 QuadTree

2.1 Tworzenie drzewa

Aby utworzyć drzewo należy utworzyć obiekt klasy QuadTree, który przyjmuje listę punktów (obiektów klasy Point lub obiektów iterowalnych) jako argument.

```
from quadtree import QuadTree

points1 = [[1, 2], [3, 4], [5, 6]]
Qtree1 = QuadTree(points1)
points2 = [Point([1, 2]), Point([3, 4])]
Qtree2 = QuadTree(points2, points_in_node=True)
points3 = [[-1, 2.23], Point([3.3, -4]), [7.3, 43.212]]
Qtree3 = QuadTree(points3, max_capacity=3)
# ValueError: The list of points is empty.
Qtree4 = QuadTree([]) # ValueError
# ValueError: The points have different dimensions.
Qtree5 = QuadTree([[1, 2], [3, 4.3, -5]]) # ValueError
```

Kod źródłowy 5: Konstruktor klasy QuadTree.

2.2 Wyszukiwanie punktu

Aby sprawdzić czy drzewo zawiera punkt należy wywołać metodę if_contains, która przyjmuje punkt (obiekt klasy Point lub obiekt iterowalny) jako argument.

```
from quadtree import QuadTree

qtree = QuadTree([[1, -2.2], [3.6, 4], Point([5, 6])])
qtree.if_contains([1, -2.2])         # True
qtree.if_contains([1, 2])            # False
qtree.if_contains(Point([3.6, 4]))   # True
qtree.if_contains(Point([3.6, 4.1])) # False
# ValueError: The point has different dimension than 2.
qtree.if_contains([1, 2, 3])         # ValueError
```

Kod źródłowy 6: Sprawdzenie czy drzewo zawiera punkt.

2.3 Wyszukiwanie punktów w prostokącie

Aby wyszukać punkty znajdujące się w prostokącie należy wywołać metodę search_in_rectangle, która przyjmuje prostokąt (obiekt klasy Rectangle) jako argument.

```
from quadtree import QuadTree

qtree = QuadTree([[1, -2.2], [3.6, 4], Point([5, 6])])
qtree.search_in_rectangle(Rectangle([0, 0], [2, 2]), raw=False) # [Point([1, -2.2])]
qtree.search_in_rectangle(Rectangle([0, 0], [3, 4]), raw=True)  # [[1, -2.2]]
qtree.search_in_rectangle(Rectangle([6, 8], [9, 9]), raw=False) # []
# ValueError: The rectanangle is not a Rectangle object.
qtree.search_in_rectangle([0, 0, 2, 2])                         # ValueError
# ValueError: The rectangle has different dimension than 2.
```

```
qtree.search_in_rectangle(Rectangle([0, 0, 4], [2, 2, 3]))      # ValueError
```

Kod źródłowy 7: Wyszukiwanie punktów w prostokącie.

3 KdTree

3.1 Tworzenie drzewa

Aby utworzyć drzewo należy utworzyć obiekt klasy KdTree, który przyjmuje listę punktów (obiektów klasy Point lub obiektów iterowalnych) jako argument.

```
from kdtree import KdTree

points1 = [[1, 2], [3, 4], [5, 6]]
Ktree1 = KdTree(points1)
points2 = [Point([1, 2]), Point([3, 4])]
Ktree2 = KdTree(points2)
points3 = [[-1, 2.23, 5], Point([3.3, -4, 2]), [7.3, 43.212, 1]]
Ktree3 = KdTree(points3)
points4 = [[1, 2], [3, 4], [5, 6], [7, 8], [9, 10]]
Ktree4 = KdTree(points4, points_in_node=True)
# ValueError: The list of points is empty.
Ktree4 = KdTree([]) # ValueError
# ValueError: The points have different dimensions.
Ktree5 = KdTree([[1, 2], [3, 4.3, -5]]) # ValueError
```

Kod źródłowy 8: Konstruktor klasy KdTree.

3.2 Wyszukiwanie punktu

Aby sprawdzić czy drzewo zawiera punkt należy wywołać metodę if_contains, która przyjmuje punkt (obiekt klasy Point lub obiekt iterowalny) jako argument.

```
ktree = KdTree([[1, -2.2], [3.6, 4], Point([5, 6])])
ktree.if_contains([1, -2.2])                # True
ktree.if_contains(Point([1, 2]))            # False
# ValueError: The point has different dimension than the points in the tree.
ktree.if_contains([1, 2, 3])                # ValueError
```

Kod źródłowy 9: Sprawdzenie czy drzewo zawiera punkt.

3.3 Wyszukiwanie punktów w prostokącie

Aby wyszukać punkty znajdujące się w prostokącie należy wywołać metodę search_in_rectangle, która przyjmuje prostokąt (obiekt klasy Rectangle) jako argument.

```
ktree = KdTree([[1, -2.2], Point([3.6, 4]), [2, -6]])
kdtree.search_in_rectangle(Rectangle([0, -3], [2, 2]), raw=False) # [Point([1, -2.2])]
ktree.search_in_rectangle(Rectangle([0, -10], [3, 2]), raw=True)  # [[1,-2.2],[2,-6]]
ktree.search_in_rectangle(Rectangle([0, 0], [2, 2]))  # []
# ValueError: The rectanangle is not a Rectangle object.
ktree.search_in_rectangle([0, 0, 2, 2])                          # ValueError
# ValueError: The rectangle has different dimension than the points in the tree.
ktree.search_in_rectangle(Rectangle([0, 0, 4], [2, 2, 3]))       # ValueError
```

Kod źródłowy 10: Wyszukiwanie punktów w prostokącie.

4 QuadTree_visualizer

Klasa umożliwiająca wizualizację drzewa QuadTree.

4.1 Wizualizacja drzewa

Aby wizualizować drzewo należy utworzyć obiekt klasy QuadTree_Visualizer, który przyjmuje listę punktów (obiektów klasy Point lub obiektów iterowalnych) jako argument oraz ewentualne modyfikatory. Zachowuje się tak jak wyżej opisana klasa QuadTree.

```
from quadtree import QuadTree_Visualizer

points1 = [[1, 2], [3, 4], [5, 6]]
# w folderze zostanie utworzony plik quadtree.gif
Qtree1 = QuadTree_Visualizer(points1, title='QuadTree', filename='quadtree')
# w folderze zostanie utworzony plik QuadTree-construction.gif
Qtree2 = QuadTree_Visualizer(points1, points_in_node=True)
```

Kod źródłowy 11: Konstruktor klasy QuadTree_Visualizer.

4.2 Wyszukiwanie punktu

Zachowuje się tak jak wyżej opisana klasa QuadTree, z tą różnicą, że zapisuje w gifie animację wyszukiwania punktu.

```
points1 = [[1, 2.2], Point([3.1, 4]), [-0.5, 6]]
Qtree1 = QuadTree_Visualizer(points1, title='QuadTree', filename='quadtree-contain')
# w folderze zostanie utworzony plik Quadtree-contains.gif
Qtree1.if_contains([1, 2.2])
```

Kod źródłowy 12: Wyszukiwanie punktu w QuadTree_Visualizer.

4.3 Wyszukiwanie punktów w prostokącie

Zachowuje się tak jak wyżej opisana klasa QuadTree, z tą różnicą, że zapisuje w gifie animację wyszukiwania punktów w prostokącie.

```
points1 = [[1, 2.2], Point([3.1, 4]), [-0.5, 6]]
Qtree1 = QuadTree_Visualizer(points1, points_in_node=True, visualize_gif=False)
# w folderze zostanie utworzony plik Quadtree-search.gif
Qtree1.search_in_rectangle(Rectangle([0, 0], [2, 2]), title='Quad1')
```

Kod źródłowy 13: Wyszukiwanie punktów w prostokącie w QuadTree_Visualizer.

5 KdTree_visualizer

Klasa umożliwiająca wizualizację operacji drzewa KdTree.

5.1 Wizualizacja drzewa

Zachowuje się tak jak wyżej opisana klasa KdTree. Przy tworzeniu obiektu zapisuje w folderze plik kdtree.gif z animacją tworzenia drzewa.

```
from kdtree import KdTree_Visualizer

points1 = [[1, 2], [3, 4], [5, 6]]
# w folderze zostanie utworzony plik kdtree.gif
Ktree1 = KdTree_Visualizer(points1, title='KdTree', filename='kdtree')
```

```
# w folderze zostanie utworzony plik kdtree-construction.gif
Ktree2 = KdTree_Visualizer(points1)
```

Kod źródłowy 14: Konstruktor klasy KdTree_Visualizer.

5.2 Wyszukiwanie punktu

Zachowuje się tak jak wyżej opisana klasa KdTree, z tą różnicą, że zapisuje w gifie animację wyszukiwania punktu.

```
points1 = [[1, 2.2], Point([3.1, 4]), [-0.5, 6]]
Ktree1 = KdTree_Visualizer(points1)
# w folderze zostanie utworzony plik kdtree-contain.gif
Ktree1.if_contains([1, 2.2], title='KdTree', filename='kdtree-contain')
```

Kod źródłowy 15: Wyszukiwanie punktu w KdTree_Visualizer.

5.3 Wyszukiwanie punktów w prostokącie

Zachowuje się tak jak wyżej opisana klasa KdTree, z tą różnicą, że zapisuje w gifie animację wyszukiwania punktów w prostokącie.

```
points1 = [[1, 2.2], Point([3.1, 4]), [-0.5, 6]]
Ktree1 = KdTree_Visualizer(points1)
# w folderze zostanie utworzony plik kdtree-search.gif
Ktree1.search_in_rectangle(Rectangle([0, 0], [2, 2]))
```

Kod źródłowy 16: Wyszukiwanie punktów w prostokącie w KdTree_Visualizer.

6 Testy

Aby przetestować strukturę należy utworzyć obiekt klasy TestManager, który przyjmuje drzewo (obiekt klasy QuadTree lub KdTree) jako argument, a następnie uruchomić interesujące nas funkcje. Rezultat testów jest wyświetlany w konsoli.

```
testman1 = TestManager(QuadTree)
testman2 = TestManager(KdTree)
testman1.all_tests()                  # wykonuje wszystkie testy integracyjne
testman2.all_unittest()               # wykonuje wszystkie testy jednostkowe
testman1.contain_point_int()          # wykonuje testy dla punktow int
testman2.contain_point_float()        # wykonuje testy dla punktow float
testman2.search_in_rectangle_int()    # wykonuje testy dla punktow int
testman2.search_in_rectangle_float() # wykonuje testy dla punktow float
```

Kod źródłowy 17: Testy.

7 Wizualizacja

Przykładowe wizualizacje znajdują się w pliku *visualizer/demo.ipynb*.

8 Porównanie

Porównanie wydajności struktur QuadTree i KdTree znajduje się w pliku *Comparision.ipynb*, wraz z generacją wykresów oraz otrzymanymi siatkami.

IV. Sprawozdanie

1 Wstęp teoretyczny

1.1 Cel sprawozdania

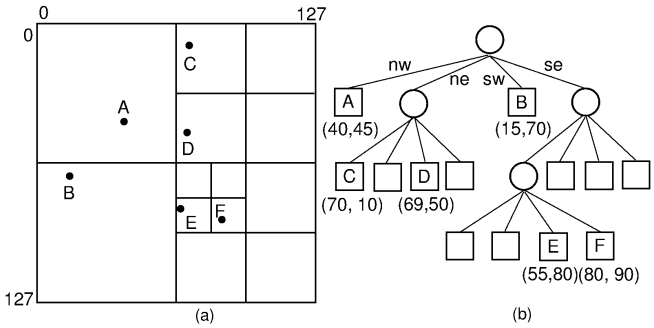
Dane są punkty na płaszczyźnie określające prostokąt $P1(x1, y1)$, oraz $P2(x2, y2)$.
Należy znaleźć zbiór punktów Q ze zbioru spełniających warunki:

- $x_1 < q_x < x_2$
- $y_1 < q_y < y_2$

Sprawozdania ma na celu rozwiązanie problemu przeszukiwania zbioru punktów, w celu znalezienia tych, które należą do zadanego przez nas obszaru. Przedstawione zostaną podstawowe informacje o strukturach QuadTree oraz KdTree, a także opis ich implementacji oraz porównanie wydajności dla zróżnicowanych danych wejściowych.

1.2 QuadTree

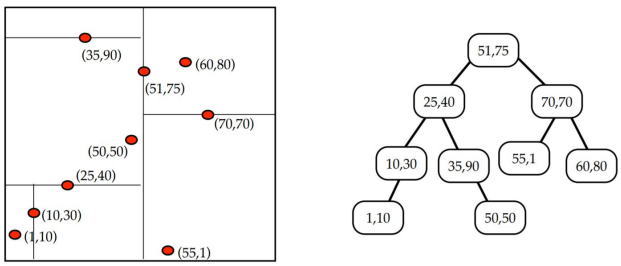
QuadTree jest strukturą danych, która umożliwia przechowywanie punktów w dwuwymiarowej przestrzeni. Służy do podziału dwuwymiarowej przestrzeni na mniejsze części, dzieląc je na cztery równe ćwiartki, a następnie podzielone ćwiartki na kolejne cztery ćwiartki, aż do osiągnięcia maksymalnej pojemności węzła. Węzeł naszego drzewa bedzie prostokątnym obszarem, który będzie reprezentował drzewo czwórkowe, a liście będą również prostokątnymi obszarami, które nie zostały podzielone ze względu na ilość punktów w danym obszarze.



Rysunek 1: Przykładowe drzewo QuadTree.

1.3 KdTree

KdTree jest strukturą danych będącą drzewem binarnym, która umożliwia przechowywanie punktów w wielowymiarowej przestrzeni. Służy do podziału wielowymiarowej przestrzeni na mniejsze części, dzieląc ją na dwie części względem punkt mediany dla kolejnych wymiarów, a następnie podzielone części na kolejne dwie części względem ich mediany, aż do momentu dotarcia do liścia, który będzie zawierał jeden punkt.



Rysunek 2: Przykładowe drzewo KdTree.

2 Opis implementacji

2.1 QuadTree

Strukturę inicjuje się poprzez utworzenie obiektu klasy QuadTree, który przyjmuje listę punktów, które mają zostać dodane do drzewa. Wtedy w kolejnych węzłach punkty są dzielone na podstawie przynależności do danej ćwiartki i z utworzonego podzbioru tworzony jest kolejny węzeł zawierający należące do niego punkty, który powtarza operacje do momentu, w którym ilość punktów nie będzie przekraczać ustalonej wartości $max\_capacity$.

W trakcie wyszukiwania zadanego punktu w drzewie, punkt przechodzi przez kolejne węzły w ten sam sposób jaki byłby dodawany do drzewa, a więc wchodzi do kolejnych węzłów, w których się zawiera, aż do momentu dotarcia do liścia, w którym następuje sprawdzenie czy punkt znajduje się w liściu.

Podczas wyszukiwania punktów w prostokącie, sprawdzane są wszystkie węzły, które przecinają się z prostokątem, a następnie rekurencyjnie dochodzimy do liści, w których sprawdzamy czy punkt znajduje się w prostokącie.

Złożoność czasowa wyszukiwania punktu w drzewie wynosi $O(dl)$, gdzie d to głębokość drzewa, a l to ilość liści w drzewie.

2.2 KdTree

Strukturę inicjuje się poprzez utworzenie obiektu klasy KdTree, który przyjmuje listę punktów, które mają zostać dodane do drzewa. Wtedy w kolejnych węzłach punkty są sortowane względem danego wymiaru i wybierana jest mediana, na podstawie której punkty są dzielone na dwie części. Z utworzonego podzbioru tworzony jest kolejny węzeł, który powtarza operacje dla następnego wymiaru, aż do momentu, w którym węzeł będzie zawierał jeden punkt.

W trakcie wyszukiwania zadanego punktu w drzewie, punkt jest porównywany z osią tworzącą węzeł, i na tej podstawie przechodzi do odpowiedniego węzła, aż do momentu dotarcia do liścia, w którym następuje sprawdzenie czy punkt znajduje się w liściu.

Podczas wyszukiwania punktów w prostokącie, sprawdzane są wszystkie węzły, które przecinają się z prostokątem, a następnie rekurencyjnie dochodzimy do liści, w których sprawdzamy czy punkt znajduje się w prostokącie.

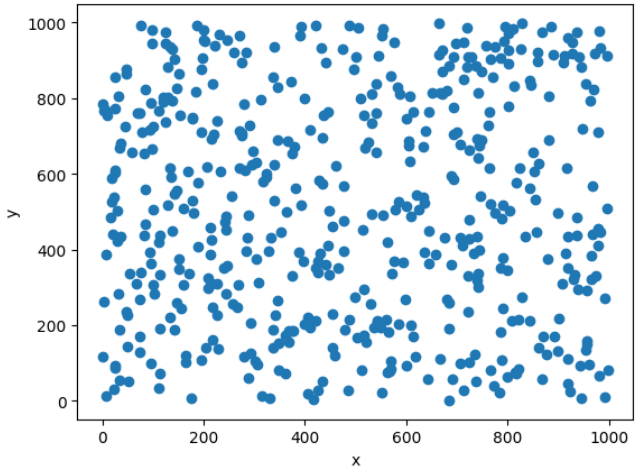
Złożoność obliczeniowa utworzenia drzewa KD wynosi $O(kn * logn)$, gdzie n to ilość punktów w drzewie, k to ilość wymiarów. Przy zrównoważonym drzewie złożoność obliczeniowa wyszukiwania wynosi $O(\sqrt{n} + k)$, gdzie n to ilość punktów w drzewie, k to ilość punktów wynikowych.

3 Porównanie wyników dla różnych danych wejściowych

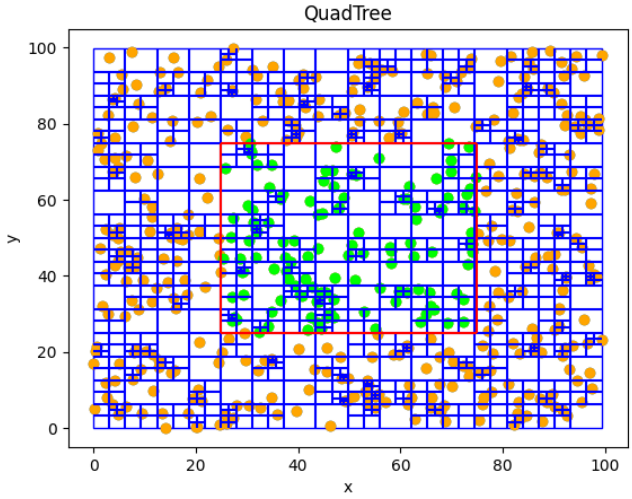
Porównania znajdują się w pliku *Comparision.ipynb*, zostały tam przedstawione wykresy oraz opis porównania wydajności struktur dla różnych danych wejściowych. wykorzystane zostały zbiory punktów wygenerowane przez klasę *CaseGenerator* w zbiorach od 10000 do 100000 punktów z krokiem 10000 wewnątrz prostokąta o wierzchołkach (0,0), (100,0), (100,100), (0,100). Porównane zostały czasy inicjalizacji struktury oraz czasy wyszukiwania punktów w strukturze.

3.1 Zbiór punktów o rozkładzie jednorodnym

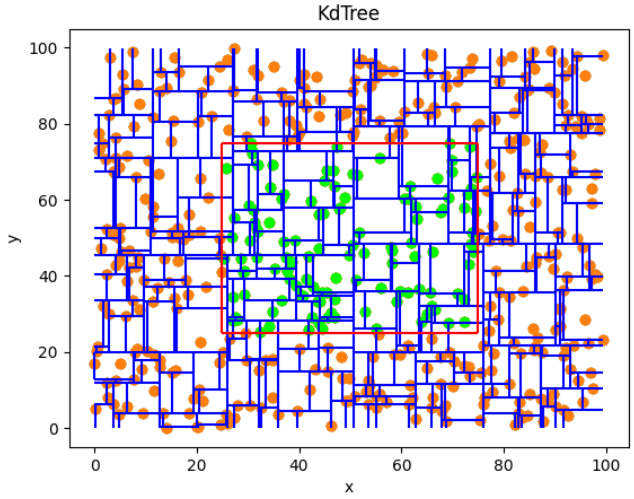
Zbiór reprezentujący losowo wygenerowane punkty w przestrzeni dwuwymiarowej należących do danego prostokąta. Wyszukujemy punkty znajdujące się w obszarze, który częściowo przecina nasz zbiór punktów. Jest to przypadek neutralny, ponieważ punkty są równomiernie rozłożone w przestrzeni.



Rysunek 3: Zbiór punktów o rozkładzie jednorodnym.



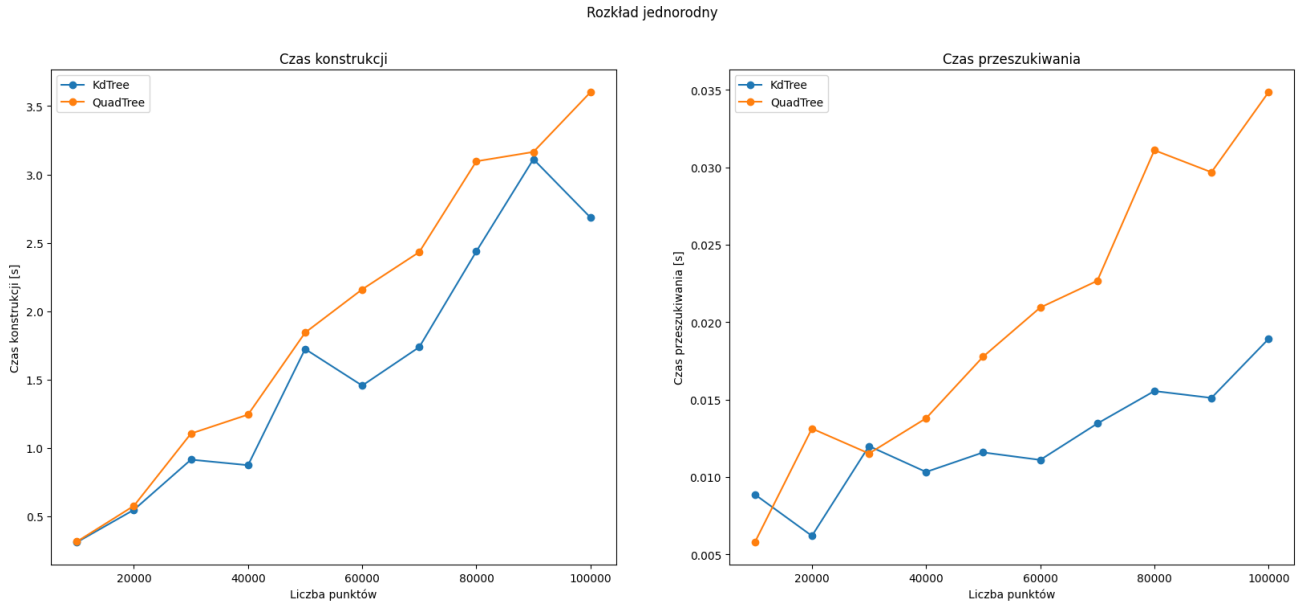
Rysunek 4: Siatka QuadTree rozkładu jednorodnego.



Rysunek 5: Siatka KdTree rozkładu jednorodnego.

		Rozkład jednorodny			
		Czas konstrukcji [s]		Czas przeszukiwania [s]	
	Liczba punktów	KdTree	QuadTree	KdTree	QuadTree
1	10000	0.525953	0.603661	0.006965	0.017502
2	20000	0.879578	1.301307	0.015709	0.031569
3	30000	1.833377	2.940357	0.035963	0.055741
4	40000	3.320225	3.408814	0.033606	0.052383
5	50000	4.035366	4.130799	0.036009	0.065693
6	60000	5.321494	5.128195	0.046706	0.071691
7	70000	6.319651	6.413196	0.056525	0.109037
8	80000	6.937868	7.912826	0.025324	0.065647
9	90000	3.682173	3.769052	0.026852	0.082187
10	100000	7.567008	8.679212	0.055804	0.140044

Tabela 2: Wyniki pomiarów porównujące wydajność QuadTree i KdTree dla zbioru rozkładu jednorodnego.

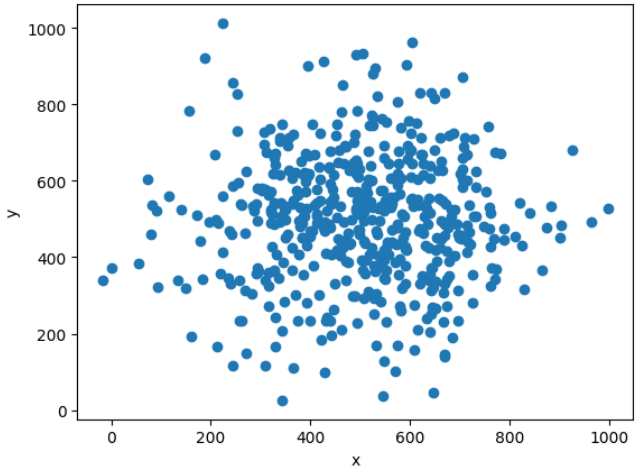


Rysunek 6: Wykres porównujący wydajność QuadTree i KdTree dla zbioru punktów o rozkładzie jednorodnym.

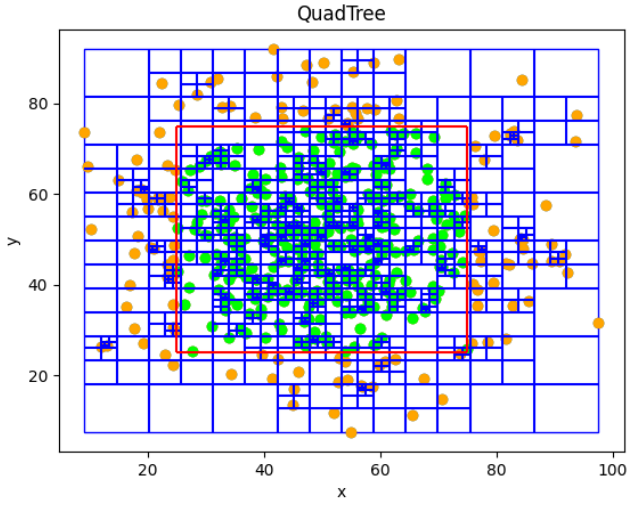
Na powyższym wykresie widać, że KdTree jest szybsze zarówno podczs konstrukcji, jak i przeszukiwania. Wraz ze wzrostem liczby punktów KdTree powiększa swoją przewagę nad QuadTree.

3.2 Zbiór punktów o rozkładzie normalnym

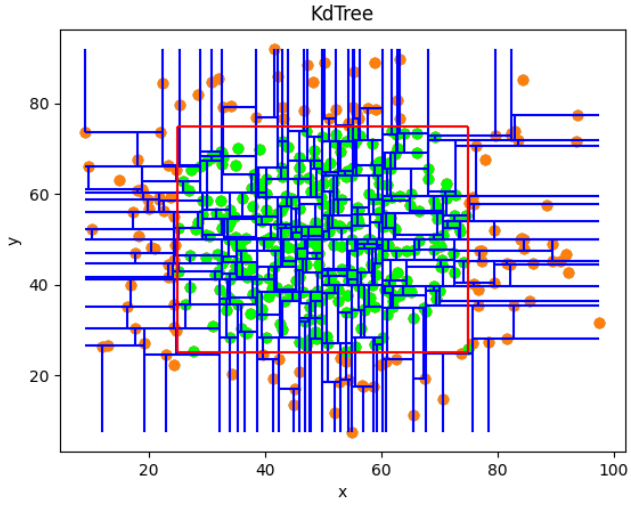
Zbiór reprezentujący punkty wygenerowane zgodnie z rozkładem normalnym w przestrzeni dwuwymiarowej. Oznacza to, że punkty są bardziej skupione w środku prostokąta, a na brzegach jest ich mniej. Przeszukiwanym prostokątem jest prostokąt o wierzchołkach (25,25), (75,25), (75,75), (25,75).



Rysunek 7: Zbiór punktów o rozkładzie normalnym.



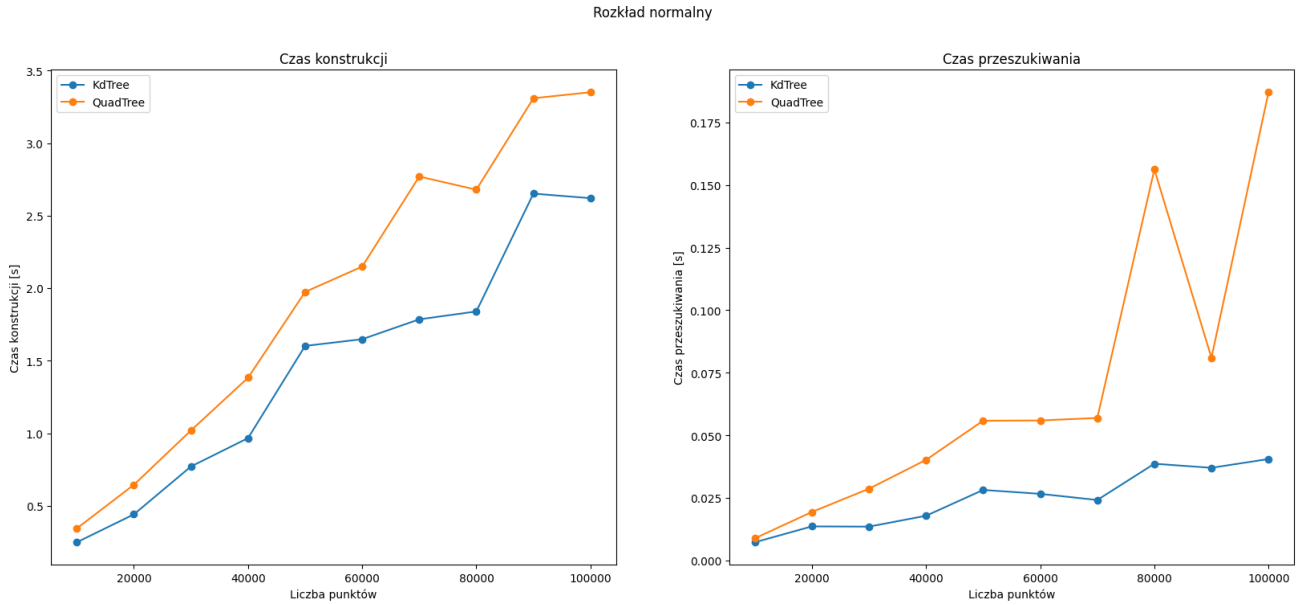
Rysunek 8: Siatka QuadTree dla rozkładu normalnego.



Rysunek 9: Siatka KdTree dla rozkładu normalnego.

		Rozkład normalny			
		Czas konstrukcji [s]		Czas przeszukiwania [s]	
	Liczba punktów	KdTree	QuadTree	KdTree	QuadTree
1	10000	0.739546	0.952264	0.027823	0.037308
2	20000	1.707641	1.774976	0.042721	0.078259
3	30000	2.229701	2.539679	0.042665	0.088242
4	40000	3.394858	3.609045	0.085345	0.124107
5	50000	4.173224	4.534599	0.063369	0.141578
6	60000	4.914829	5.526659	0.079745	0.199600
7	70000	6.192708	6.572183	0.091050	0.247608
8	80000	6.833023	7.734797	0.100284	0.349896
9	90000	7.688269	7.967488	0.103521	0.321787
10	100000	9.013417	9.272826	0.113198	0.334062

Tabela 3: Wyniki pomiarów porównujące wydajność QuadTree i KdTree dla rozkładu normalnego.

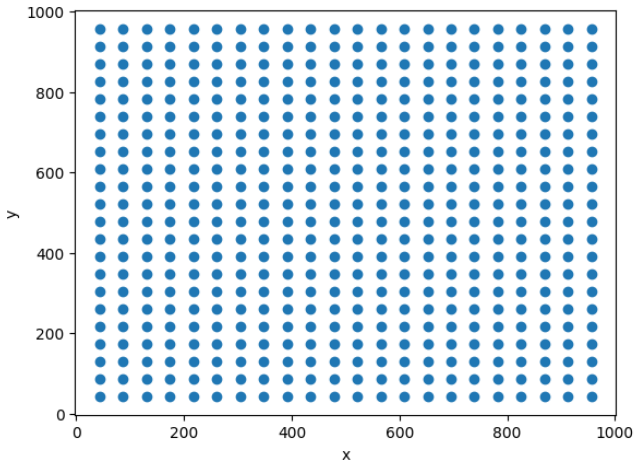


Rysunek 10: Wykres porównujący wydajność QuadTree i KdTree dla zbioru punktów o rozkładzie normalnym.

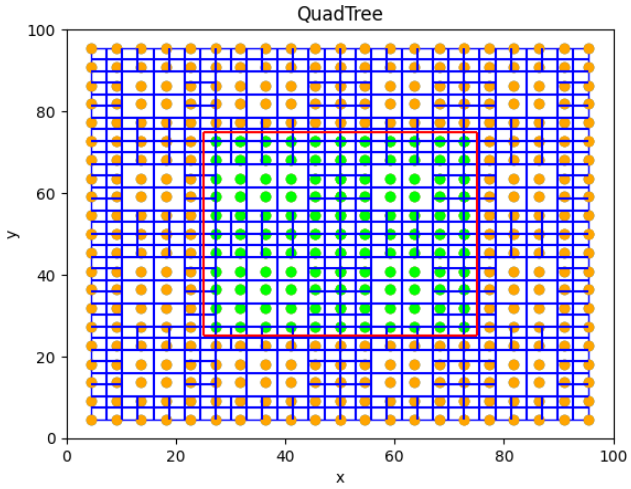
Na powyższym wykresie widać, że tak jak poprzednio czas inicjalizacji struktur jest niemal identyczny, oraz poczas przeszukiwania KdTree radzi sobie lepiej względem QuadTree. Zauważalne są spore niestabliności przy największych zbiorach (pomimo zastosowania średniego czasu trzech wykonań).

3.3 Zbiór punktów o rozkładzie 'siatka'

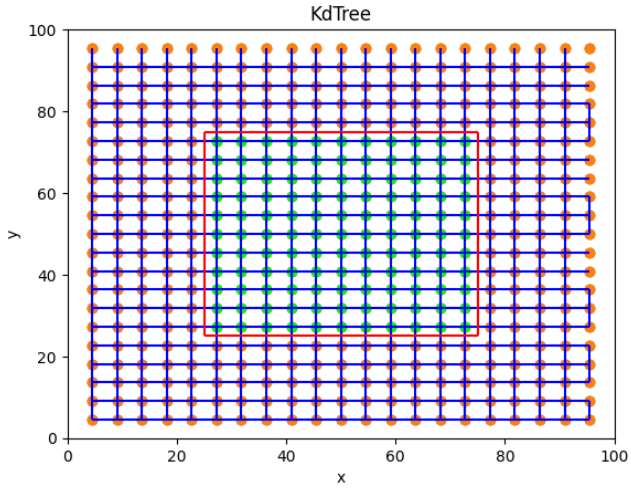
Rozkład punktów przypomina siatkę, ponieważ punkty są równomiernie rozłożone wzdłuż osi X i Y wewnątrz zadanego prostokąta. Przeszukiwanym prostokąt jest prostokąt o wierzchołkach (25,25), (75,25), (75,75), (25,75). Zbiór ten jest problematyczny, ponieważ może zawierać wiele punktów na osi podziału, co może spowodować, że drzewo będzie niezrównoważone.



Rysunek 11: Zbiór punktów o rozkładzie "siatka".



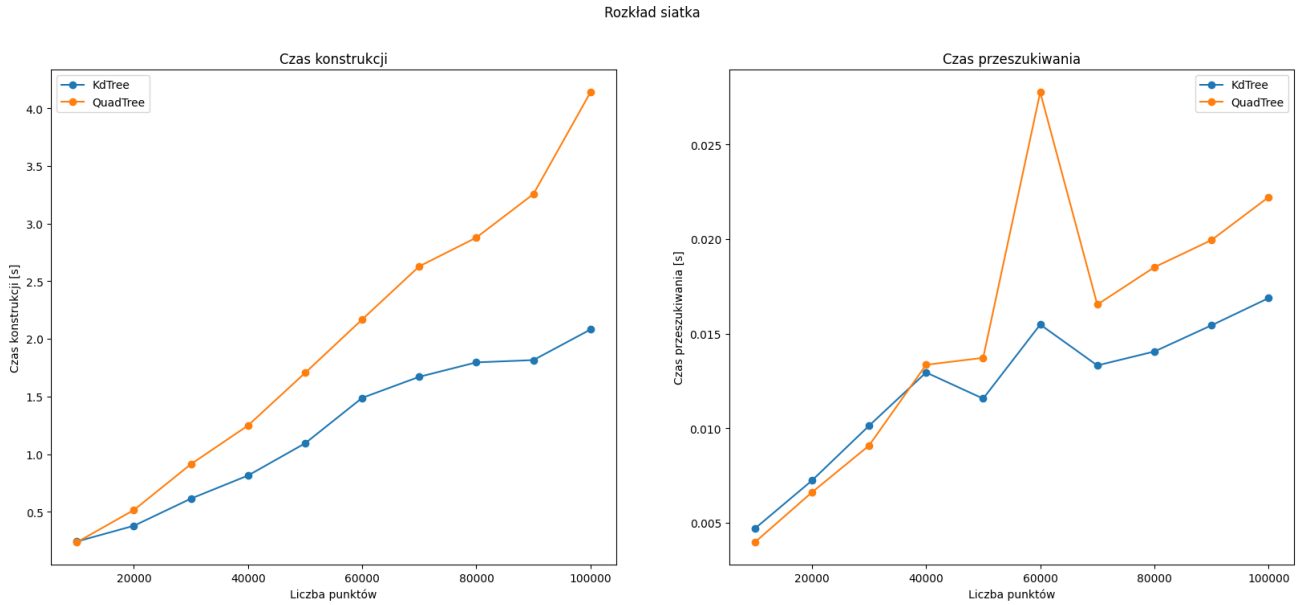
Rysunek 12: Siatka QuadTree dla rozkładu "siatka".



Rysunek 13: Siatka KdTree dla rozkładu "siatka".

		Rozkład siatka			
		Czas konstrukcji [s]		Czas przeszukiwania [s]	
	Liczba punktów	KdTree	QuadTree	KdTree	QuadTree
1	10000	0.760563	0.804475	0.017787	0.020623
2	20000	1.191172	1.315219	0.027098	0.030674
3	30000	1.830970	2.464700	0.027581	0.038512
4	40000	2.300369	3.230727	0.055203	0.071101
5	50000	3.731767	4.215484	0.038001	0.068557
6	60000	4.130582	5.627054	0.046119	0.053191
7	70000	4.493137	6.378847	0.057497	0.077872
8	80000	5.352753	7.442259	0.047987	0.083077
9	90000	5.616267	8.432021	0.049193	0.079609
10	100000	6.655318	9.113047	0.062163	0.091143

Tabela 4: Wyniki pomiarów porównujące wydajność QuadTree i KdTree dla rozkładu "siatka".

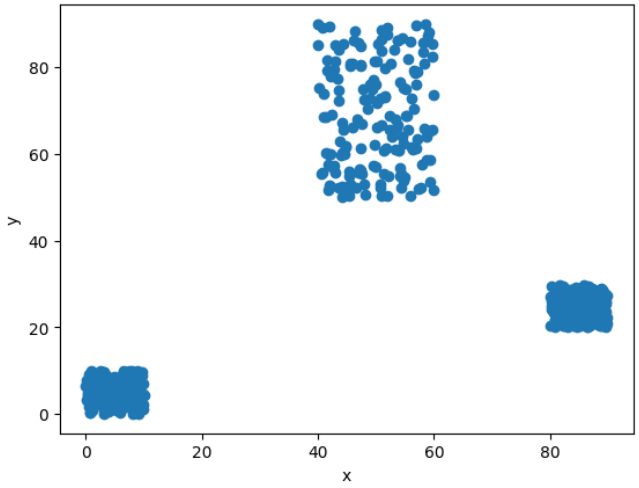


Rysunek 14: Wykres porównujący wydajność QuadTree i KdTree dla zbioru punktów o rozkładzie "siatka".

W tym przypadku ponownie widać zbliżony czas zarówno inicjalizacji struktur, jaki i przeszukiwania. Jednak oo połowy zakresu porównania drzewo ćwiartkowe nie dotrzymuje tempa KdTree.

3.4 Zbiór punktów o rozkładzie klastrowym

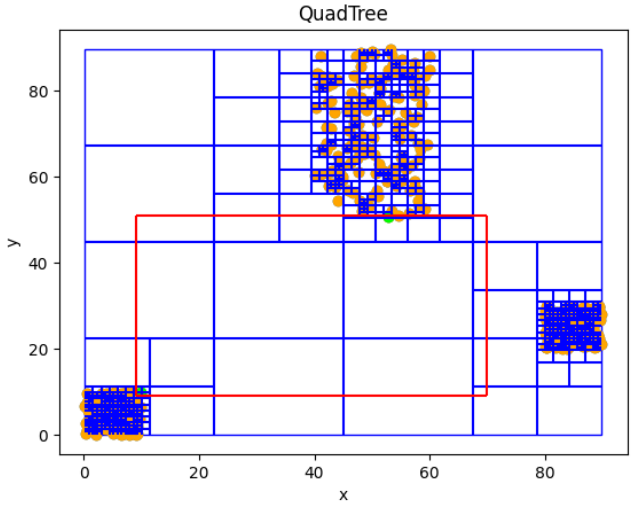
Ten przypadek reprezentuje trzy grupy mocno skupionych punktów w przestrzeni dwuwymiarowej. Ze względu na niezrównoważone wybory prostokąta zedecydowano się na podział tego problemu na dwa przypadki, aby zobaczyć jak struktury zachowują się w odmiennych sytuacjach.



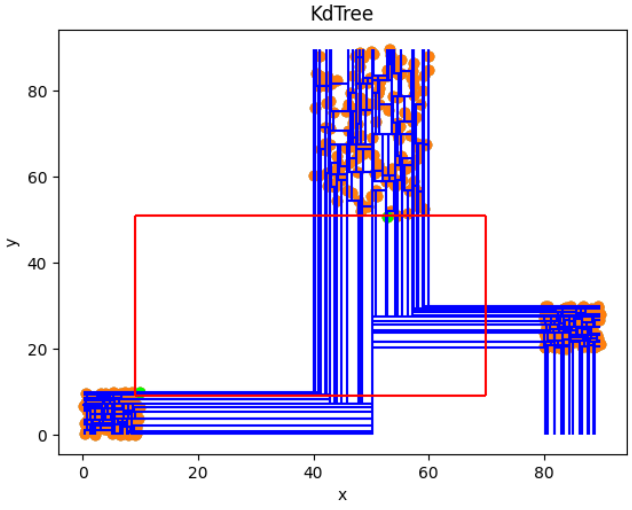
Rysunek 15: Zbiór punktów o rozkładzie klastrowym.

3.4.1 Z małą ilością punktów (1)

Dla tego przypadku obasz zawiera nieznaczne fragmenty dwóch skupisk punktów, i dominuje znaczna pusta przestrzeń,



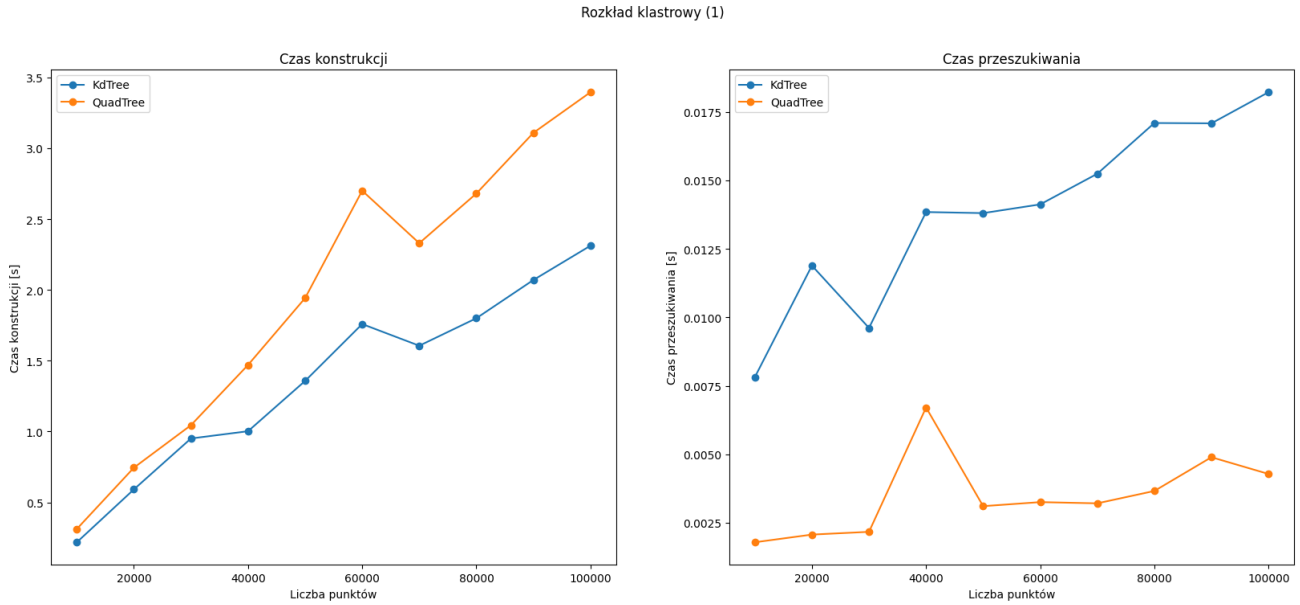
Rysunek 16: Siatka QuadTree dla rozkładu klaster (1).



Rysunek 17: Siatka KdTree dla rozkładu klaster (1).

	Rozkład klastrowy (1)				
		Czas konstrukcji [s]		Czas przeszukiwania [s]	
	Liczba punktów	KdTree	QuadTree	KdTree	QuadTree
1	10000	0.744485	1.146938	0.019429	0.004075
2	20000	1.384678	1.626412	0.045429	0.007739
3	30000	2.832320	2.930801	0.034926	0.008536
4	40000	3.370394	4.088355	0.044638	0.009340
5	50000	4.737663	3.483118	0.026201	0.008898
6	60000	6.363855	6.455460	0.054979	0.011443
7	70000	5.795496	6.191663	0.086012	0.012112
8	80000	6.720413	6.898166	0.058796	0.010809
9	90000	7.459068	7.749592	0.061746	0.014384
10	100000	13.345561	6.145183	0.054499	0.008764

Tabela 5: Wyniki pomiarów porównujące wydajność QuadTree i KdTree dla rozkładu klaster (1).



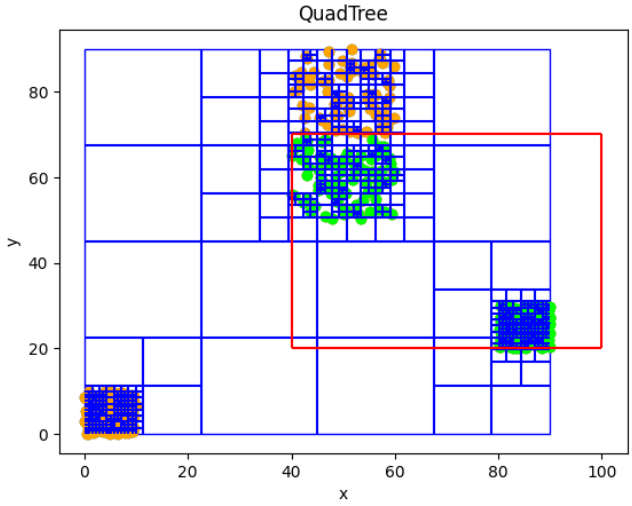
Rysunek 18: Wykres porównujący wydajność QuadTree i KdTree dla rozkładu klaster (1).

Czas konstrukcji drzew jest zbliżony dla tej samej liczby punktów.

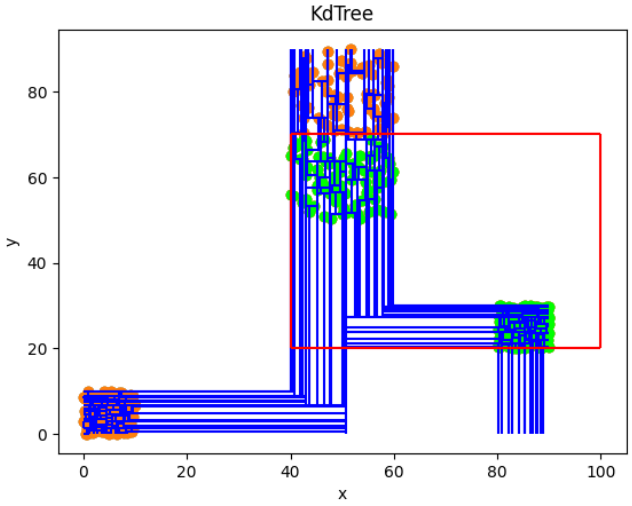
Podczas przeszukiwania QuadTree osiągało zbliżone czasy, mimo większej ilości punktów (nielicząc pojedynczego odchylenia), co oznacza, że sprawniej radziło sobie z przechodzeniem po odpowiednich węzłach.

3.4.2 Z dużą ilością punktów (2)

Dla tego przypadku przeszukujemy obszar zawierający znaczne fragmenty dwóch skupisk punktów.



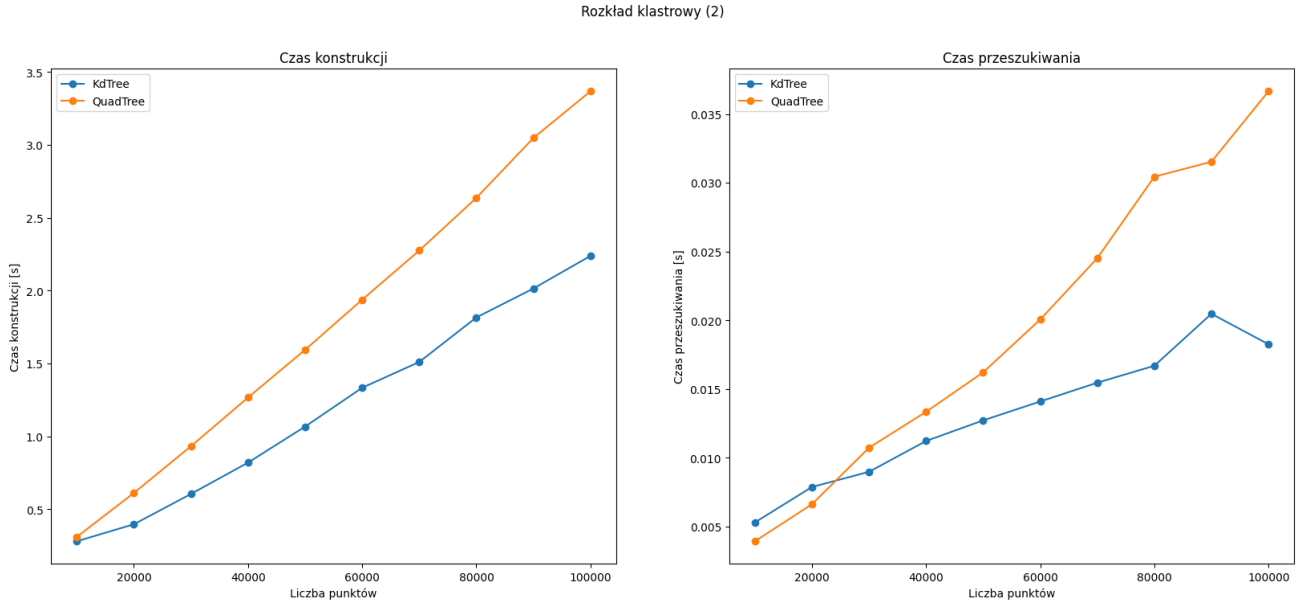
Rysunek 19: Siatka QuadTree dla rozkładu klastrowego (2).



Rysunek 20: Siatka KdTree dla rozkładu klastrowego (2).

		Rozkład klastrowy (2)			
		Czas konstrukcji [s]		Czas przeszukiwania [s]	
	Liczba punktów	KdTree	QuadTree	KdTree	QuadTree
1	10000	0.332947	0.379944	0.011154	0.009497
2	20000	0.796904	0.860786	0.037727	0.051893
3	30000	1.849791	1.823151	0.023719	0.033841
4	40000	1.731005	1.727538	0.019790	0.028582
5	50000	2.212551	2.230030	0.020901	0.036624
6	60000	2.680116	2.618285	0.037243	0.091985
7	70000	3.694794	3.129360	0.095646	0.086734
8	80000	3.436740	3.804738	0.036856	0.069029
9	90000	4.169493	4.244927	0.033543	0.078159
10	100000	4.692992	4.678534	0.033104	0.101984

Tabela 6: Wyniki pomiarów porównujące wydajność QuadTree i KdTree dla rozkładu klastrowego (2).



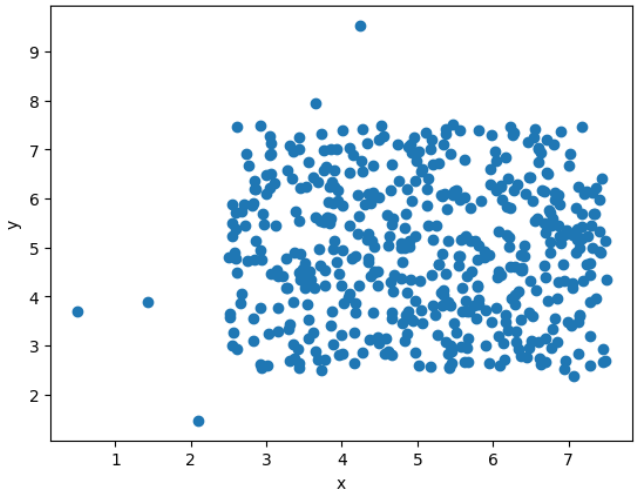
Rysunek 21: Wykres porównujący wydajność QuadTree i KdTree dla rozkładu klastrowego (2).

W tym przypadku KdTree potrzebowało ponownie odrobinę mniej czasu na zbudownie pełnego drzewa, zarówno jak i przy przeszukiwaniu zadanych obszarów, to ono poradziło sobie lepiej.

3.5 Zbiór punktów o rozkładzie z punktami odstającym

W tym przypadku mamy podobną sytuację co w rozkładzie normalnym, jednakże centrum prostokąta jest znacznie bardziej zagęszczone, a pojedyncze punkty znajdują się na obrzeżach prostokąta.

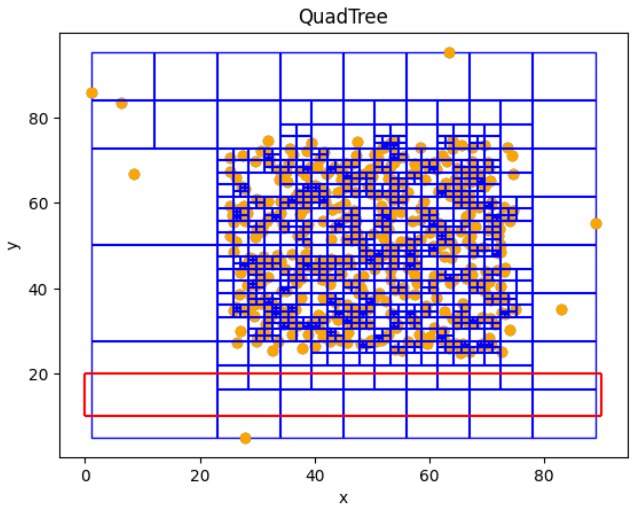
Również i tutaj zdecydowano się na podział tego problemu na dwa przypadki, aby zobaczyć jak struktury zachowują się w odmiennych sytuacjach.



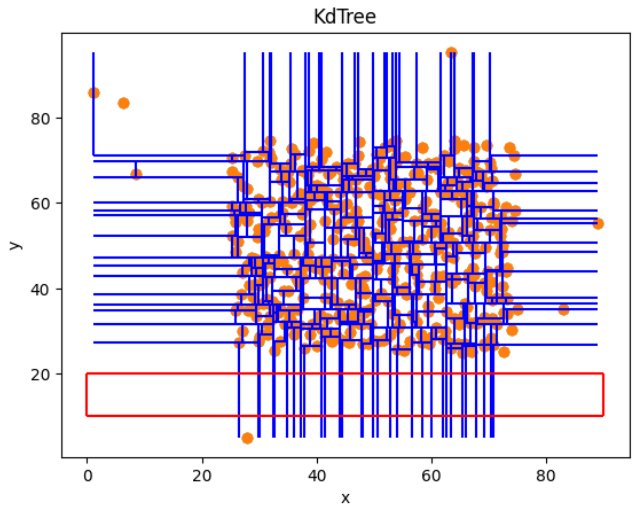
Rysunek 22: Zbiór punktów o rozkładzie z punktami odstającymi.

3.5.1 Z małą ilością punktów

W tej sytuacji przeszukiwany prostokąt nie zawiera centralnej grupy punktów, a jedynie wychwytuje pojedyncze punkty odstające.



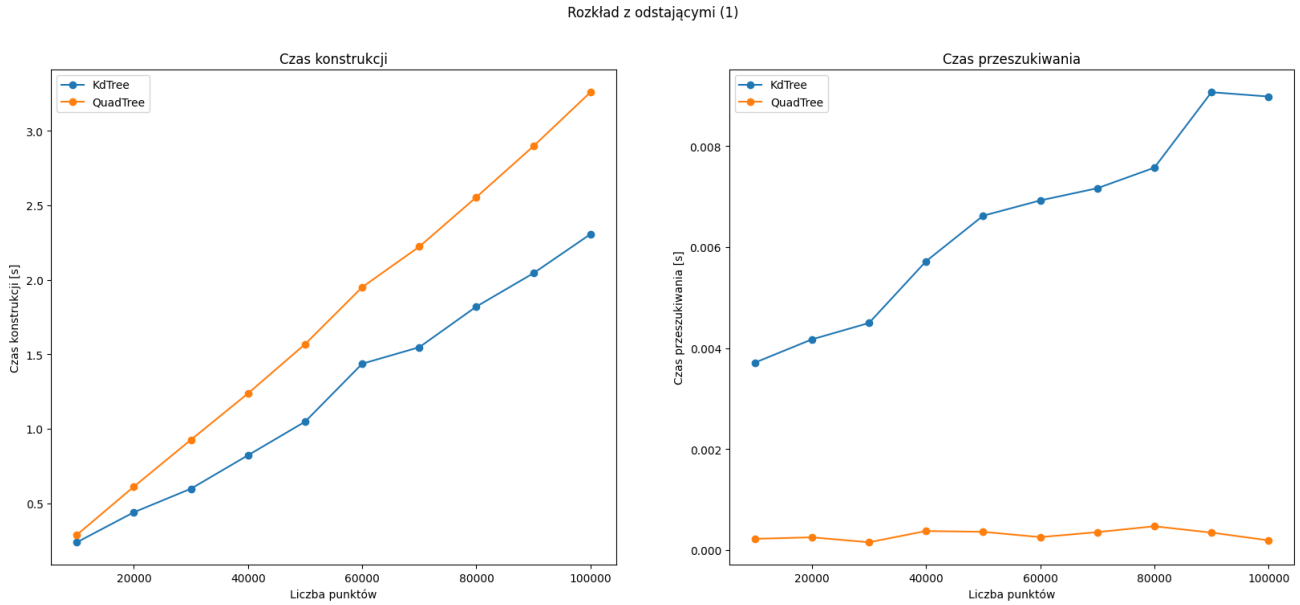
Rysunek 23: Siatka QuadTree dla rozkładu odstającego (1).



Rysunek 24: Siatka KdTree dla rozkładu odstającego (1).

	Rozkład z odstającymi (1)				
		Czas konstrukcji [s]		Czas przeszukiwania [s]	
	Liczba punktów	KdTree	QuadTree	KdTree	QuadTree
1	10000	0.489375	0.524964	0.004784	0.000502
2	20000	1.154447	1.115454	0.014153	0.000350
3	30000	1.546494	1.476078	0.009427	0.000544
4	40000	1.533275	1.927339	0.014799	0.000809
5	50000	2.083945	2.201441	0.010279	0.000359
6	60000	2.967560	2.735626	0.022038	0.000294
7	70000	3.082076	3.363270	0.013432	0.000267
8	80000	3.217827	5.419607	0.018007	0.000354
9	90000	4.329930	4.179824	0.020457	0.000221
10	100000	4.419229	4.760538	0.018829	0.000348

Tabela 7: Wyniki pomiarów porównujące wydajność QuadTree i KdTree rozkładu odstającego (1).

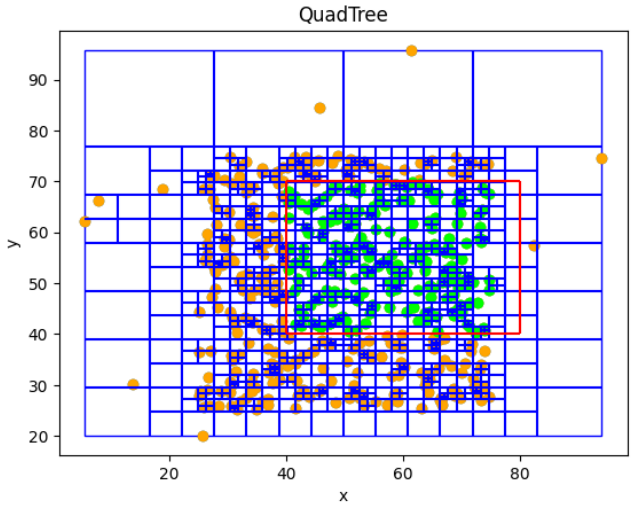


Rysunek 25: Wykres porównujący wydajność QuadTree i KdTree dla rozkładu odstającego (1).

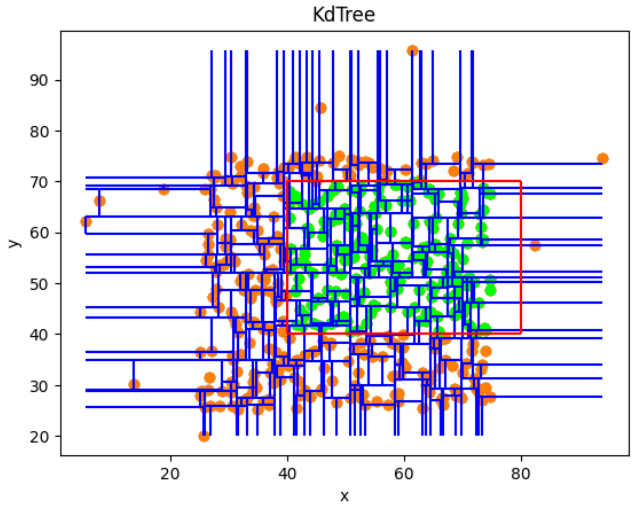
Budowa obu drzew zajmuje mniej więcej tyle samo czasu z małą przewagą dla struktury KdTree. Czas przeszukiwania, jednak znacznie przeważa na korzyść drzewa ćwiartkowego.

3.5.2 Z dużą ilością punktów

W tej sytuacji przeszukiwany obszar zawiera znaczną część centralnej grupy punktów, a także pojedyncze punkty odstające.



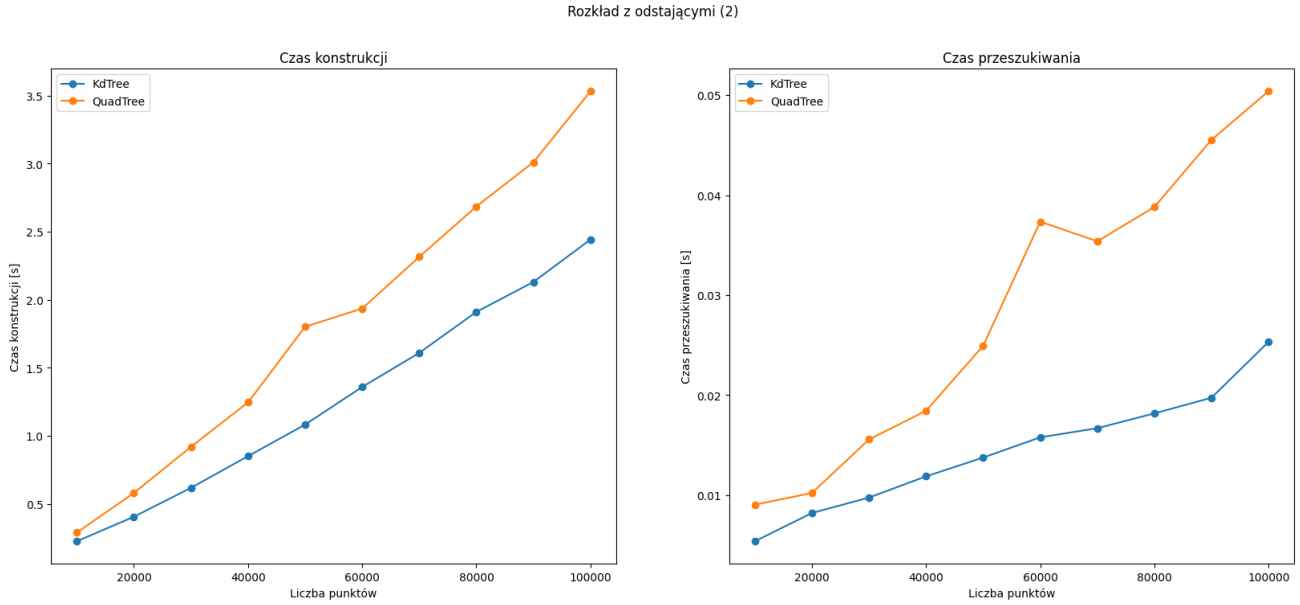
Rysunek 26: Siatka QuadTree dla rozkładu odstającego (2).



Rysunek 27: Siatka KdTree dla rozkładu odstającego (2).

		Rozkład z odstającymi (2)			
		Czas konstrukcji [s]		Czas przeszukiwania [s]	
	Liczba punktów	KdTree	QuadTree	KdTree	QuadTree
1	10000	0.392502	0.392262	0.019904	0.035270
2	20000	0.725716	0.889906	0.014218	0.022862
3	30000	1.438910	1.690897	0.015602	0.028825
4	40000	1.543269	1.709909	0.036825	0.069247
5	50000	3.925621	3.695152	0.042901	0.101671
6	60000	5.170620	6.477736	0.025804	0.074978
7	70000	3.204646	3.253412	0.032528	0.075253
8	80000	3.257927	3.435762	0.034024	0.090642
9	90000	3.725386	3.910516	0.039311	0.109010
10	100000	4.184204	4.359409	0.041051	0.125713

Tabela 8: Wyniki pomiarów porównujące wydajność QuadTree i KdTree dla rozkładu odstającego (2).



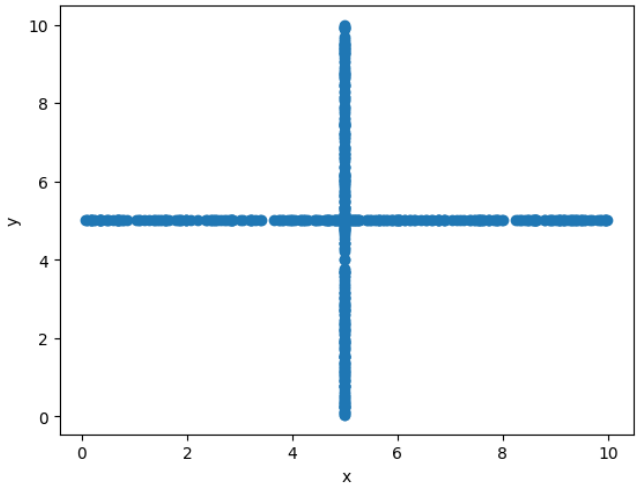
Rysunek 28: Wykres porównujący wydajność QuadTree i KdTree dla rozkładu odstającego (2).

Z wykresu możemy wywnioskować, że czasy budowy obu drzew nie różnią się zbytnio od siebie. Czas przeszukiwania natomiast jest szybszy przy użyciu Kd-drzewa. Wykres przypomina nieco wykresy z początkowych podpunktów.

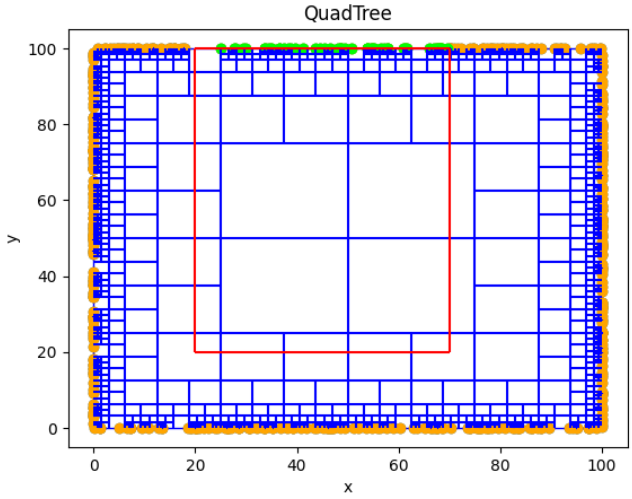
3.6 Zbiór punktów o rozkładzie krzyżowym

W tym przypadku mamy podobną sytuację co w rozkładzie "siatka", jednakże punkty są równomiernie rozłożone wzdłuż osi symetrii prostokąta, a więc znacznie więcej punktów jest współliniowych.

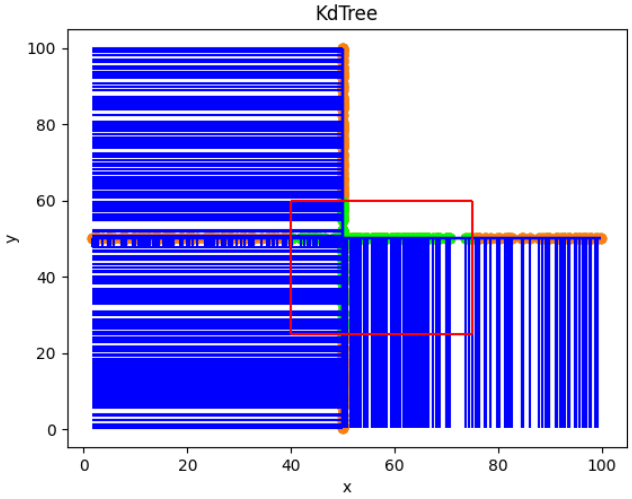
Przeszukiwany obszar zawiera fragmenty obu prostych, co powoduje, że drzewo będzie musiało pokonać znacznie większą głębokość, aby znaleźć wszystkie punkty.



Rysunek 29: Zbiór punktów o rozkładzie krzyżowym.



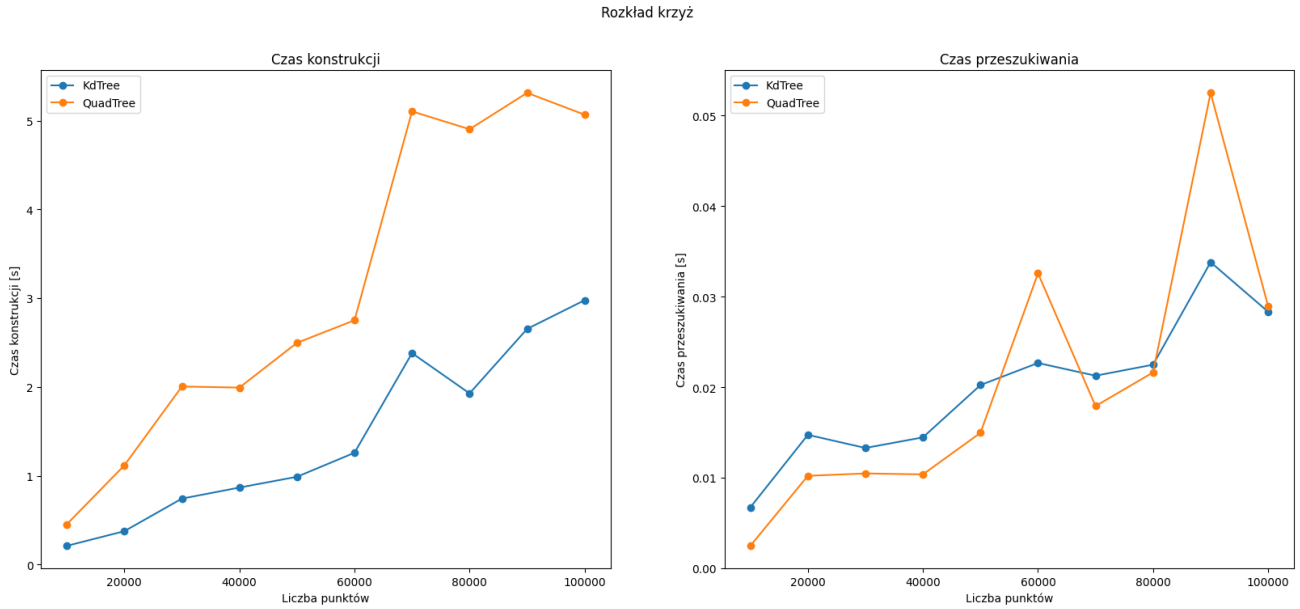
Rysunek 30: Siatka QuadTree dla rozkładu krzyżowego.



Rysunek 31: Siatka KdTree dla rozkładu krzyżowego.

		Rozkład krzyż			
		Czas konstrukcji [s]		Czas przeszukiwania [s]	
	Liczba punktów	KdTree	QuadTree	KdTree	QuadTree
1	10000	0.352320	0.828095	0.019195	0.010147
2	20000	0.932730	1.604570	0.027632	0.013407
3	30000	1.251809	4.130071	0.056319	0.048069
4	40000	2.510517	6.102843	0.045348	0.029465
5	50000	3.323393	7.474479	0.025906	0.031376
6	60000	1.975971	5.418630	0.033230	0.042832
7	70000	2.326260	4.737657	0.042741	0.037731
8	80000	2.736304	5.495954	0.038266	0.049964
9	90000	3.156827	6.273638	0.044632	0.052923
10	100000	3.848184	9.214811	0.054329	0.067162

Tabela 9: Wyniki pomiarów porównujące wydajność QuadTree i KdTree dla rozkładu krzyżowego.

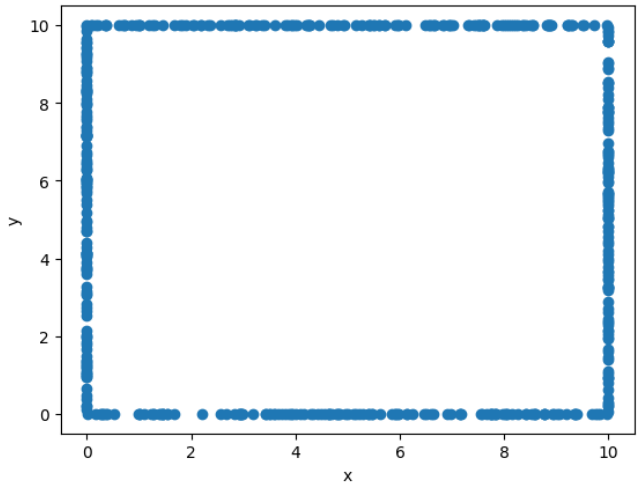


Rysunek 32: Wykres porównujący wydajność QuadTree i KdTree dla rozkładu krzyżowego.

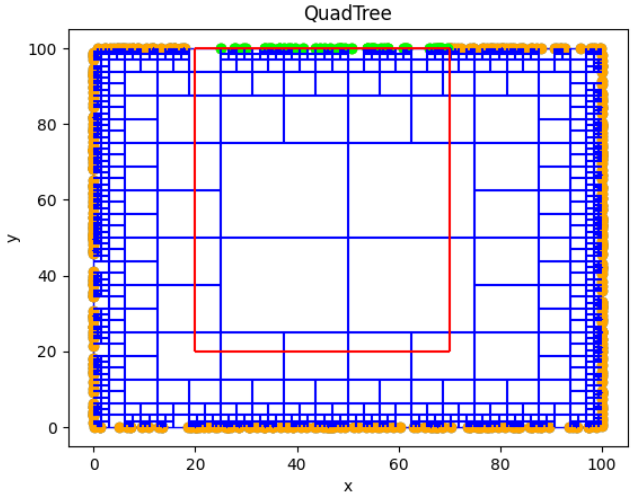
Czas budowy QuadTree jest dużo wolniejszy w przypadku zbioru Krzyż. Iwdać również skoki czasu podczas przeszukiwania zbiorów, co udowadnia wrażliwość tej struktury na zbiór danych. KdTree dużo szybciej się buduje, a jego czas wyszukiwania jest dużo stabilniejszy.

3.7 Zbiór punktów o rozkładzie na bokach prostokąta

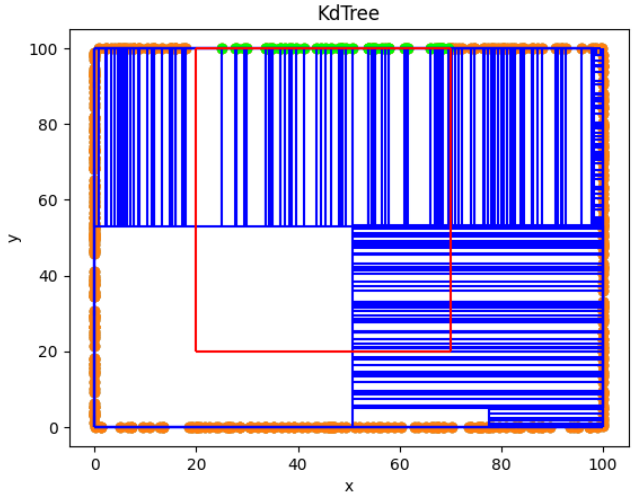
Sytuacja podobna do poprzedniej, jednakże zagęszczamy problem w większej ilości miejsc.



Rysunek 33: Zbiór punktów o rozkładzie prostokątnym.



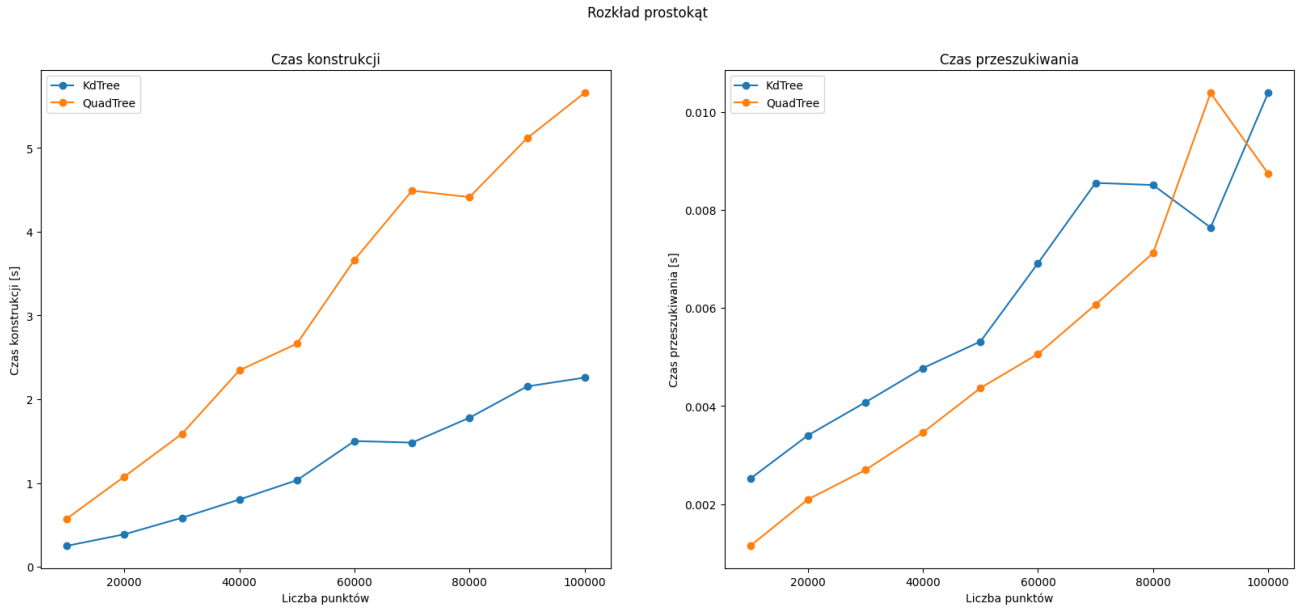
Rysunek 34: Siatka QuadTree dla zbioru punktów o rozkładzie prostokątnym.



Rysunek 35: Siatka KdTree dla zbioru punktów o rozkładzie prostokątnym.

	Rozkład prostokąt				
		Czas konstrukcji [s]		Czas przeszukiwania [s]	
	Liczba punktów	KdTree	QuadTree	KdTree	QuadTree
1	10000	0.375075	0.649978	0.007469	0.002012
2	20000	0.610409	1.282686	0.005383	0.003395
3	30000	0.946753	2.131414	0.017567	0.015938
4	40000	1.923439	3.270247	0.012884	0.010967
5	50000	3.759927	7.306431	0.017963	0.015527
6	60000	4.311115	8.518162	0.028830	0.021768
7	70000	3.606635	6.099763	0.025063	0.029072
8	80000	7.144494	12.368523	0.017309	0.018718
9	90000	3.574068	10.166259	0.022601	0.031538
10	100000	7.407704	12.718501	0.022318	0.026166

Tabela 10: Wyniki pomiarów porównujące wydajność QuadTree i KdTree dla zbioru punktów o rozkładzie na bokach prostokąta.



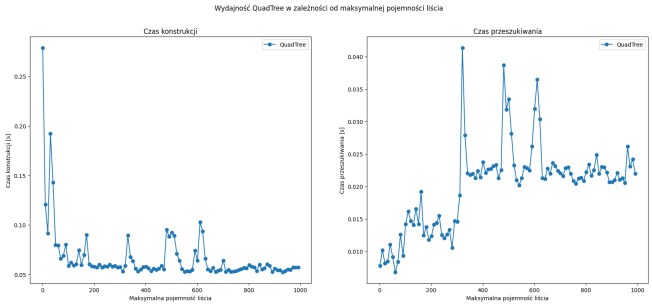
Rysunek 36: Wykres porównujący wydajność QuadTree i KdTree dla zbioru punktów o rozkładzie prostokątnym.

Czas konstrukcji w tym przypadku jest dużo większy dla QuadTree, okazuje się, że nie radzi on sobie dobrze z danymi rozłożonymi gęsto na linii. Czas przeszukiwania jest nieco wolniejszy przy KdTree z jedną różnicą w zbiorze 9.

3.8 Porównanie efektywności QuadTree dla różnych wartości max_capacity

Przeprowadzono testy dla różnych wartości max_capacity, aby sprawdzić jak wpływa ona na czas konstrukcji i przeszukiwania QuadTree.

Porównane zostały operacje dla zbioru o rozkładzie normalnym liczącego 5000 punktów. Natomiast wartości capacity zawierały się w przedziale od 1 do 1000.



Rysunek 37: Wykres porównujący wydajność QuadTree dla różnych wartości max_capacity.

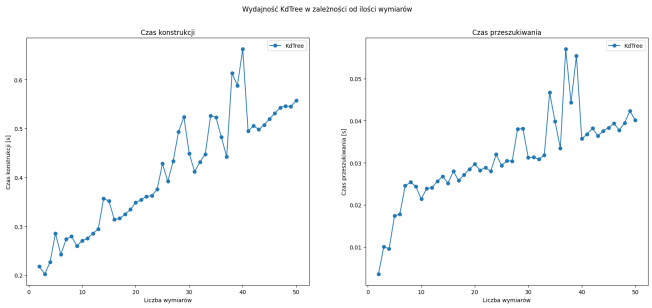
Czas konstrukcji dla max_capacity mniejszych niż 200 gwałtownie spada, następnie wykres jest bardziej płaski. Wynika to z faktu, że dla małych wartości max_capacity, drzewo jest znacznie głębsze, a więc musi wykonać znacznie więcej operacji, aby znaleźć odpowiednią komórkę. Dla większych wartości max_capacity, drzewo jest znacznie płytsze, a więc czas konstrukcji jest znacznie krótszy.

Odwrotną sytuację spotykamy przy przeszukiwaniu. Dla małych wartości max_capacity, drzewo jest w stanie sprawniej operować wyborem potrzebnych węzłów, natomiast dla rosnącej liczby max_capacity przeszukiwanie staje się przechodzeniem przez listę.

3.9 Testowanie KdTree dla różnej ilości wymiarów

Struktura KDTree zapewnia możliwość przechowywania punktów dla różnych wymiarów. Stąd właśnie i pochodzi nazwa - KD od "k dimensions".

Porównane zostały operacje dla zbioru o rozkładzie jednorodnym liczącego 1000 punktów. Natomiast wartości wymiarów zawierały się w przedziale od 2 do 50.



Rysunek 38: Wykres porównujący wydajność KdTree dla różnych ilości wymiarów.

Z powyższego wykresu widać, że czas konstrukcji i przeszukiwania gwałtownie rośnie, aż do momentu przekroczenia 10 wymiarów. Po tym momencie czas przeszukiwania zwalnia, jednakże mimo zastosowania wartości średniej z 3 pomiarów, wciąż widać znaczne niestabilne wartości. Jedną z przyczyn jest losowe generowanie punktów, które może powodować, że w niektórych przypadkach punkty będą się znajdować w jednym obszarze, a w innych w zupełnie innym, jednakże można zauważyć pewnego rodzaju trend rosnący.

Czyli widzimy, że ilość wymiarów wpływa wprost na czas przeszukiwania. I dla małej ilości punktów przy dużej ilości wymiarów nie warto używać struktury KDTree.

4 Podsumowanie

Na podstawie testów przeprowadzonych dla wyżej przeanalizowanych zbiorów, możemy stwierdzić, że programy przez nas zaimplementowane działają prawidłowo i poprawnie wyznaczają podzbiór punktów należących do zadanej płaszczyzny.

Dla każdego z testowanych przypadków, czas konstruowania struktur był podobny, a więc możemy założyć poprawną implementację obu struktur.

Jednak dla ważnego konspektu jakim jest wyszukiwanie punktów można określić, że dla zbiorów gęsto oraz średnio usianych, takich jak rozkłady jednolite, czy normalne to KdTree będzie mocniejszym wyborem.

Sytuacja ma się z goła odmiennie dla zbiorów rzadkich oraz, gdy wiele punktów jest współliniowych,to drzewo ćwiartkowe wiedzie swój prym. Dzięki swojej strukturze, szybciej odcina puste węzły i zapewnia lepsze czasy.

Mimo wszystko, gdy rozkład z jakim będziemy mieli do czynienia jest nieznany to lepszym wyborem będzie drzewo Kd.

5 Bibliografia

Implementacja poszczególnych struktur danych:

- QuadTree - Weronika Wojtas
- KdTree - Radosław Rolka

Źródła i inspiracje wykorzystane przy tworzeniu projektu:

- Wykłady z Algorytmów Geometrycznych, prowadzone przez dr inż. Barbarę Głut, na 3 semestrze Informatyki AGH WI.
- https://github.com/aghbit/Algorytmy-Geometryczne
- https://en.wikipedia.org/wiki/Quadtree
- https://en.wikipedia.org/wiki/K-d_tree
- https://en.wikipedia.org/wiki/Nearest_neighbor_search
- https://en.wikipedia.org/wiki/Range_searching
- https://www.agh.edu.pl/o-agh/multimedia/znak-graficzny-agh/
- https://github.com/Goader/KDTree_QuadTree/tree/main (przypadki testowe)