Algorytmy Geometryczne

Wyszukiwanie geometryczne
KD-Trees i Quad-Trees

Marek Małek, Jakub Kotara

Gr. 4, czwartek A: 11:20 - 12:50

Contents

Część I
Dane techniczne

- System operacyjny: Windows 11 Home
- Procesor: Intel Core i5-12450H 12th Gen 2.00 GHz
- Architektura: Alder Lake-H
- Pamięć RAM: 16 GB
- Język: Python 3.11.6
- Użyte biblioteki: **numpy**, **matplotlib**, **tkinter**, **jsonpickle**, **random**, **os**, **math**, **threading**

Część II
Dokumentacja

1 Oznaczenia

- **np** - przyjęta skrócona nazwa biblioteki numpy
- **plt** - przyjęta skrócona nazwa pakietu pyplot biblioteki numpy

2 Geometry

Pakiet **Geometry** zawiera podstawowe figury geometryczne przydatne do zaimplementowania algorytmów.

2.1 Point

Klasa **Point** odpowiada za reprezentację punktów. Oprócz przechowywania informacji na temat współrzędnych punktu, posiada dodatkowe atrybuty oraz metody.
Atrybuty:

- **data** - współrzędne punktu przechowywane w formie krotki
- **dim** - liczba wymiarów przestrzeni w jakiej określony jest dany punkt. Wartość wyliczana jest na podstawie rozmiaru pola **data**

Metody:

- **__init__(data)** - kontruktor tworzący nowy obiekt klasy **Point** przyjmujący krotkę **data**
- **__eq__()** - zwraca **True**, jeżeli oba obiekty klasy **Point** posiadają identyczną liczbę wymiarów oraz wartości w każdym wymiarze
- **__hash__()** - zapewnia możliwość haszowania obiektami klasy Point
- **x()** - zwraca wartość punktu dla pierwszego wymiaru
- **y()** - zwraca wartość punktu dla drugiego wymiaru (jeżeli istnieje, wpp zwraca błąd)
- **get_dim(i)** - zwraca wartość dla i-tego wymiaru, **i** jest liczbą całkowitą typu int. Jeżeli punkt nie posiada i wymiarów, zwraca błąd

- **precedes(other)** - przyjmuje obiekt klasy Point. Sprawdza, czy wartości we wszystkich wymiarach są niewiększe niż odpowiadające im wartości obiektu **other**
- **follows(other)** - przyjmuje obiekt klasy Point. Sprawdza czy wartości we wszystkich wymiarach są niemniejsze niż odpowiadające im wartości obiektu **other**
- **lowerLeft(other)** - przyjmuje obiekt klasy Point posiadający tę samą liczbę wymiarów. Zwraca obiekt tej samej klasy posiadający minimalną wartość obu punktów we wszystkich wymiarach.
- **upperRight(other)** - przyjmuje obiekt klasy Point posiadający tę samą liczbę wymiarów. Zwraca obiekt tej samej klasy posiadający maksymalną wartość obu punktów we wszystkich wymiarach.

2.2 Rectangle

Klasa **Rectangle** odpowiada za reprezentację prostokątów / przedziałów wielowymiarowych / hiperprostokątów.
Atrybuty:

- **lowerLeft** - obiekt klasy Point, lewy dolny punkt obszaru.
- **upperRight** - obiekt klasy Point, prawy górny punkt obszaru
- **dim** - liczba wymiarów przestrzeni w jakiej określony jest hiperprostokąt

Metody:

- **__init__(point1, point2)** - przyjmuje dwa obiekty klasy Point i na ich podstawie wyznacza najmniejszy przedział, w którym oba punkty się zawierają. Na ich podstawie określa również wymiar przestrzeni **dim**
- **__str__()** - zapewnia ładne wypisywanie obiektu
- **intersects(other)** - przyjmuje obiekt klasy Rectangle. Zwraca True, jeżeli część wspólna danego przedziału wielowymiarowego oraz przekazanego jest niepustym zbiorem
- **containsPoint(point)** - przyjmuje obiekt klasy Point. Zwraca True, jeżeli punkt należy do obszaru.
- **containsRect(other)** - przyjmuje obiekt klasy Rectangle. Zwraca True jeżeli przekazany obszar całkowicie zawiera się w danym
- **divideRectIntoTwo(dim,divLine)** - przyjmuje dwa argumenty: dim - typu int, divLine - typu double. Zwraca dwa obiekty klasy Rectangle, będącymi wynikami podziału danego obszaru względem wartości **divLine** wymiarze **dim**

3 Application

Aplikacja została przygotowana przy wykorzystaniu bibliotek: **tkinter**, **matlplotlib**, **jsonpickle**, **time**.

3.1 Funkcjonalność aplikacji

Aplikacja umożliwia:

- własnoręczne zadanie testowego zbioru punktów
- własnoręczne zadanie testowego prostokąta
- zapisanie własnego testu do zbiorczego pliku **json.tests**
- wczytanie zapisanego testu

- uruchomienie wizualizacji opartej o dane drzewo oraz zbiór testowy, przy uprzednim wybraniu wariantu drzewa
- prześledzenie krokowej wizualizacji algorytmów budowania drzewa oraz wyszukiwania punktów w zadanym prostokącie dla wybranego testu

3.2 Struktura aplikacji

Aplikacja składa się z 7 modułów zaimplementowanych własnoręcznie:

- moduł **Controller** - zawiera klasę **Controller** reprezentującą kontroler między widokiem, a modelem aplikacji oraz klasę **visualisationParamers** reprezentującą parametry wizualizacji.
- moduł **Geometry** - zawiera klasy **Rectangle** i **Point**
- moduł **KDTree** - zawiera implementację KD-drzewa
- moduł **QuadTree** - zawiera implementację drzewa ćwiartkowego
- moduł **Visualiser** - zawiera klasę **Visualiser** służącą do wizualizacji obiektów i drzew
- moduł **Widgets** zawiera klasy:
  - **ExitButton** - przycisk do wychodzenia z okna
  - **Graph** - widget reprezentujący wykres bibloteki **matplotlib**
  - **Interface** - widget reprezentujący interfejs aplikacji
  - **PointOptions** - widget służący do własnoręcznego zadawania zbioru punktów
  - **RectangleOptions** - widget służący do własnoręcznego zadawania prostokąta
  - **TestFrame** - widget służący do wyboru zapisanego zbioru testowego
  - **TestName** - widget słuązacy do nadania nazwy własnemu zbiorowi testowegmu
  - **TestOptions** - widget służący jako pojemnik na szczegółowe specyfikacje własnoręcznie zadawanego testu
  - **TestFrame** - widget służący do wyboru typu drzewa na potrzeby wizualizacji
  - **VisualisationInfo** - widget opisujący parametry wizualizacji
- moduł **windows** - zawiera klasę **Application** reprezentującą główne okno aplikacji, klasę **RandomWindow** reprezentującą okno służące do zadawania zakresów dla losowań punktów lub prostokątów oraz klasę **VisualisationWindow** reprezentującą okno wizualizacji.

4 KD-Tree

Moduł **KD-Tree** zawiera dwie klasy: **kdTree** oraz **kdNode** reprezentujące k-wymiarowe drzewo pozwalające na przeszukiwanie geometryczne w szybkim czasie. Jego częstszym zastosowaniem jest znajdywanie najbliższego punktu do zadanego, ale ta funkcja nie została zaimplementowała, gdyż nie jej dotyczył projekt.

4.1 kdNode

Węzeł kdDrzewa zawierający informacje na temat podziałów punktów.
Atrybuty:

- **left** - wskazanie na następny obiekt klasy **kdNode** reprezentujący punkty większe względem szczególnego wymiaru od danego węzła

- **right** - wskazanie na następny obiekt klasy **kdNode** reprezentujący punkty większe względem szczególnego wymiaru od danego węzła
- **axis** - wartość współrzędnej w szczególnym wymiarze, względem której następuje przecięcie przestrzeni
- **rect** - obiekt klasy **Rectangle**, obszar zawierający wszystkie punkty należące do danego węzła
- **dim** - wymiar przestrzeni względem której następuje przecięcie
- **children** - lista zawierająca obiekty klsay **Point**, wszystkie punkty należące do węzła

Metody:

- **allLeaves()** - zwraca listę **children**
- **countLeaves()** - zwraca liczbę punktów należących do węzła

4.2 kdTree

Atrybuty:

- **dim** - wymiar podprzestrzeni w jakiej określone są wszystkie punkty. Jeśli punkty są określone w różnowymiarowych przestrzeniach zwraca błąd.
- **root** - obiekt klasy **kdNode**, korzeń drzewa

Metody:

- **__init__(points)** - przyjmuje listę obiektów klasy **Point**. Określa wartość **dim** oraz buduje drzewo
- **buildTree(points)** - przymuje listę obiektów klasy **Point**. W sposób rekurencyjny tworzy kolejne węzły i buduje całe drzewo.
- **search(rectangle)** - przyjmuje obiekt klasy **Rectangle**. Zwraca wszystkie punkty zawierające się w przekazanym przedziale wielowymiarowym.
- **countKD(rectangle)** - przyjmuje obiekt klasy **Rectangle**. Zwraca liczbę punktów zawierających się w przekazanym przedziale wielowymiarowym.
- **draw(vis)** - przyjmuje obiekt klasy **Visualizer**, na którym ma zwizualizować strukturę drzewa.

W module **KD-Tree** zaimplementowana została dodatkowo funkcja **QuickSelect()**, przyjmująca listę obiektów klasy **Point** oraz granice przedziału, z którego wybiera się medianę względem danego wymiaru przestrzeni w taki sposób, że wszystkie punkty na lewo od punktu-mediany mają mniejszą wartość w danym wymiarze, a na prawo większą.

5 Quad-Tree

Moduł **QuadTree** zawiera dwie klasy: **quadTree** oraz **quadTreeNode**, reprezentujące drzewo czwórkowe pozwalające na przeszukiwanie geometryczne dwuwymiarowej przestrzeni w szybkim czasie.

5.1 quadTreeNode

Węzeł drzewa czwórkowego zawierający informacje na temat podziałów punktów.
Atrybuty:

- **boundary** - obiekt klasy **Rectangle** reprezentujący prostokąt
- **capacity** - liczba punktów należących do obszaru **boundary**
- **points** - lista obiektów klasy **Point**, punkty należące do obszaru **Boundary**

- **northWest** - obiekt klasy **quadTreeNode**, lewa górna ćwiartka **boundary**
- **northEast** - obiekt klasy **quadTreeNode**, prawa górna ćwiartka **boundary**
- **southWest** - obiekt klasy **quadTreeNode**, lewa dolna ćwiartka **boundary**
- **southEast** - obiekt klasy **quadTreeNode**, prawa dolna ćwiartka **boundary**
- **isLeaf** - wartość boolowska wskazująca, czy dany węzeł jest liściem drzewa

Metody:

- **__init__(capacity, boundary)** - konstruktor tworzący nowy obiekt w oparciu o podane parametry
- **insert(point)** - wstawia punkt do drzewa, dokonuje podziału przez wywołanie metody **__divide()** jeśli jest taka potrzeba
- **__divide()** - metoda prywatna, dzieli obszar **boundary** na cztery ćwiartki tworząc dla każdej ćwiartki nowy węzeł
- **search(rect)** - przyjmuje obiekt klasy **Rectangle**, zwraca zbiór punktów zawierających się w zadanym prostokącie
- **draw(vis)** - przyjmuje obiekt klasy **Visualizer**, na którym ma zwizualizować strukturę węzła drzewa, z którego jest wywoływana oraz jego dzieci. Wizualizuje ten węzeł oraz rekruencyjnie jego dzieci.

5.2 quadTree

Atrybuty:

- **maxPoints** - maksymalna liczba punktów należąca do obszaru reprezentowanego przez węzeł **quadTreeNode** będacy jednocześnie liściem
- **root** - obiekt klasy **quadTreeNode**, korzeń drzewa

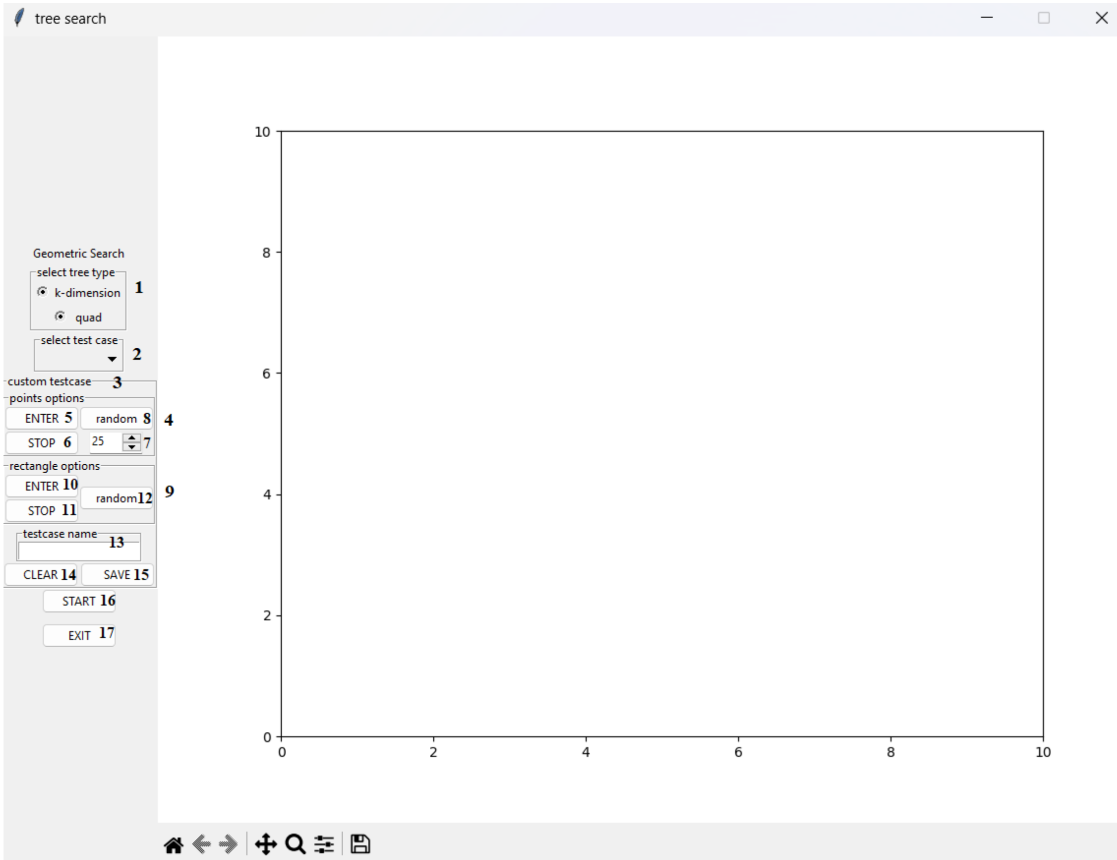
Metody:

- **__init__(points, maxPoints)** - przyjmuje listę obiektów klasy **Point** oraz liczbę naturalną. Na podstawie podanych parametrów buduje drzewo.
- **draw(vis)** - przyjmuje obiekt klasy **Visualizer**, na którym ma zwizualizować strukturę drzewa. Wizualizuje tę strukturę.
- **__findBorders(points)** - przyjmuje listę obiektów klasy **Point**. Zwraca obiekt klasy **Rectangle**, najmniejszy prostokąt zawierający wszystkie punkty
- **__buildTree(points)** - przyjmuje listę obiektów klasy **Point** funkcja budująca drzewo w rekurencyjny sposób
- **search(rect)** - przyjmuje obiekt klasy **Rectangle**. Zwraca listę obiektów klasy **Point** (punkty zawierające się w zadanym prostokącie)

Część III
Użycie programu

1 Aplikacja

Aby mieć pewność co do funkcjonalności programu należy zainstolwać wyżej wymienione biblioteki. W celu uruchomienia aplikacji należy wykonać skrypt **main.py** znajdujący się w folderze **application**.

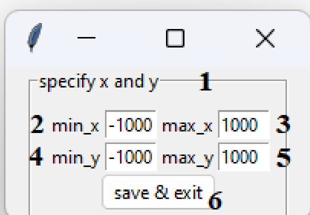


Wizualizacja 1: Główne okno aplikacji

Po lewej stronie umieszczony jest interfejs aplikacji (elementy korespondujące z wizualizacją oznaczono numerem w nawiasach):

- (1) pole **select tree type** służy do wyboru typu drzewa poprzez zaznaczenie jednej z dwóch opcji
- (2) pole **select test case** służy do wczytania zbioru testowego z pliku **tests.json**
- (3) pole **custom test case** służy do zadania własnego zbioru testowego:
  - (4) pole **point options** służy do specyfikacji zbioru punktów:
    - (5) przycisk **ENTER** uruchuamia możliwość zadawania punktów na wykresie po prawej stronie
    - (6) przycisk **STOP** zatrzymuje możliwość zadawania punktów
    - (7) wartość w spinboxie pod przyciskiem **RANDOM** określa liczbę punktów, które zostaną wygenerowane losowo po wciśnięciu przycisku
    - (8) przcisk **RANDOM** wyświetla okno służące do określenia przedziałów dla x i y oraz generuje zbiór punktów na wykresie po prawej stronie. **UWAGA!** - zadane zakresy muszą być prawidłowe: $min\_x < max\_x \wedge min\_y < max\_y$ wpp. program ostrzeże użytkownika i nakaże wprowadzić poprawne zakresy danych.
  - (9) pole **rectangle options** służy do specyfikacji prostokąta, w którym będą wyszukiwane punkty w wizualizacji.
    - (10) przycisk **ENTER** służy do wprowadzenia prostokąta na wykresie po prawej stronie
    - (11) przycisk **STOP** służy do zaprzestania wprowadzania prostokąta

    * (12) przycisk **RANDOM** służy do zadania przedziałów dla x i y oraz generuje losowy prostokąt z tego zakresu. Podobnie jak w przypadku zakresów generowania losowych punktów program przeprowadza walidację wpisywanych wartości.
  - (13) pole **test case name** posiada pole tekstowe, w którym należy wpisać nazwę dla własnego testu. **UWAGA!** - w przypadku gdy użytkownik chce stworzyć drugi test o tej samej nazwie nastąpi nadpisanie starych danych.
  - (14) przycisk **CLEAR** służy do wyczyszczenia wykresu po prawej stronie
  - (15) przycisk **SAVE** służy do zapisu własnego testu. **UWAGA!** - aby zapisać test, potrzebny jest zadany zbiór punktowy, prostokąt oraz nazwa wpp. program ostrzeże użytkownika.
- (16) przycisk **START** otwiera nowe okno przeznaczone do wizualizacji wybranego zbioru testowego oraz dla wybranego typu drzewa. **UWAGA!** - aby otworzyć okno wizualizacja musi posiadać komplet parametrów: typ drzewa, nazwę, zbiór punktów oraz prostokąt, wpp. program ostrzeże użytkownika.
- (17) przycisk **EXIT** służy do zamknięcia programu, zalecane jest aby korzystać z niego w celu poprawnego zamyknięca wszystkich widgetów.



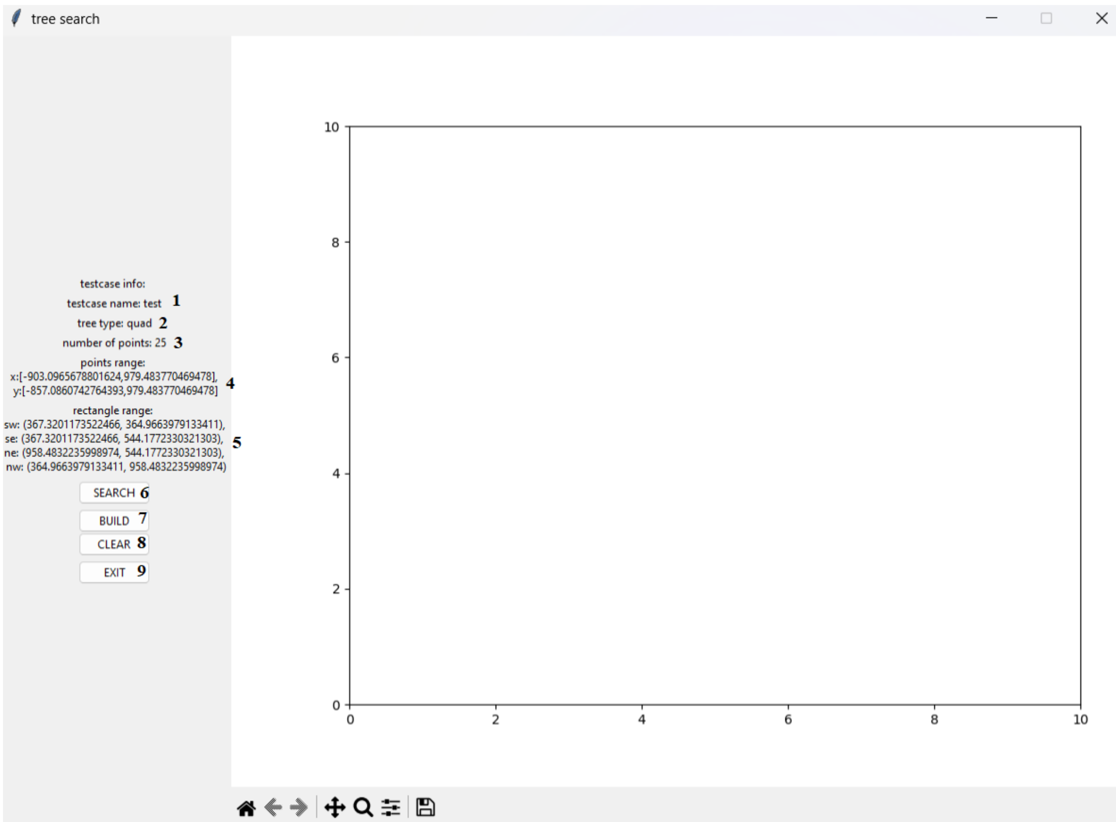
Wizualizacja 2: Okno specyfikacji zakresów

Okno specyfikacji zakresów składa się z:

- (1) - pola **specify x and y**, w którym zawarte są widgety
- (2) - pola teksowego razem z opisem **min_x**, w którym należy wpisać dolną granicę dla parametru x
- (3) - pola tekstowego razem z opisem **max_x**, w którym należy wpisać górną granicę dla parametru x
- (4) - pola teksowego razem z opisem **min_y**, w którym należy wpisać dolną granicę dla parametru y

- (5) - pola tekstowego razem z opisem **max_y**, w którym należy wpisać górną granicę dla parametru y
- (6) - przycisku do zapisania parametrów oraz zamknięcia okna specyfikacji

Przycisk **START**, oznaczony numerem 16 na wizualizacji głównego okna aplikacji (**Wizualizacja 1**) otwiera nowe okno dla zadanych parametrów wizualizacji.



Wizualizacja 3: Okno wizualizacji

Okno wizualizacji składa się z intefejsu umieszczonego po lewej stronie oraz wykresu umieszczonego po prawej stronie. W interfejsie możemy wyróżnić następujące elementy:

- (1) nazwa testu
- (2) typ drzewa wybranego do testu
- (3) liczba punktów w zbiorze testowym
- (4) zakres, z którego pochodzą punkty
- (5) cztery wierzchołki prostokąta podawane kolejno w kolejności przeciwnej do ruchu wskazówek zegara zaczynając od **sw** - dolny lewy, następnie **se**, dolny prawy, **ne** - górny prawy oraz **nw** - górny lewy.
- (6) przycisk uruchamiający wizualizację krokową wyszukiwania punktów w zadanym prostokącie.
- (7) przycisk uruchamiający wizualizację budowania drzewa
- (8) przycisk przyśpieszający wizualizację w celu tzw. przwinięcia jej
- (9) przycisk zamykający okno wizualizacji

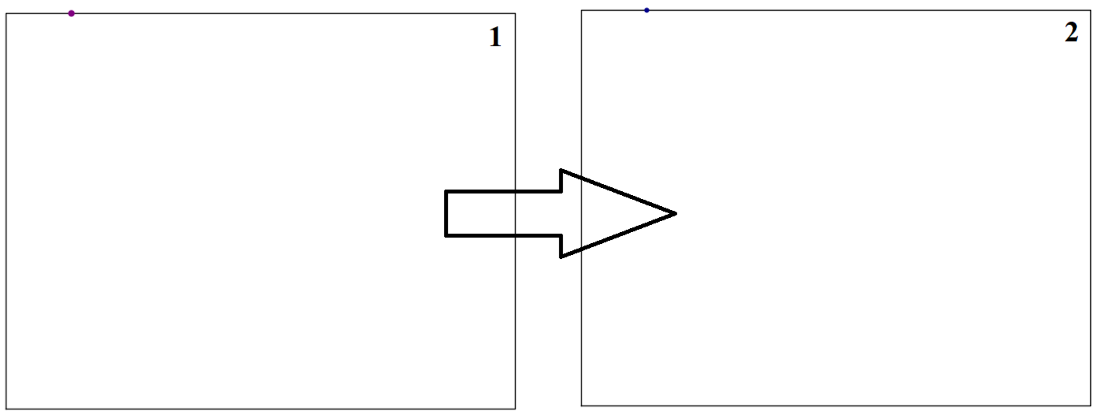
2 Wizualizacja

2.1 Quad-Tree

Omówienie przykładowej krokowej wizualizacji przyjmuje **capacity** drzewa jako 1.

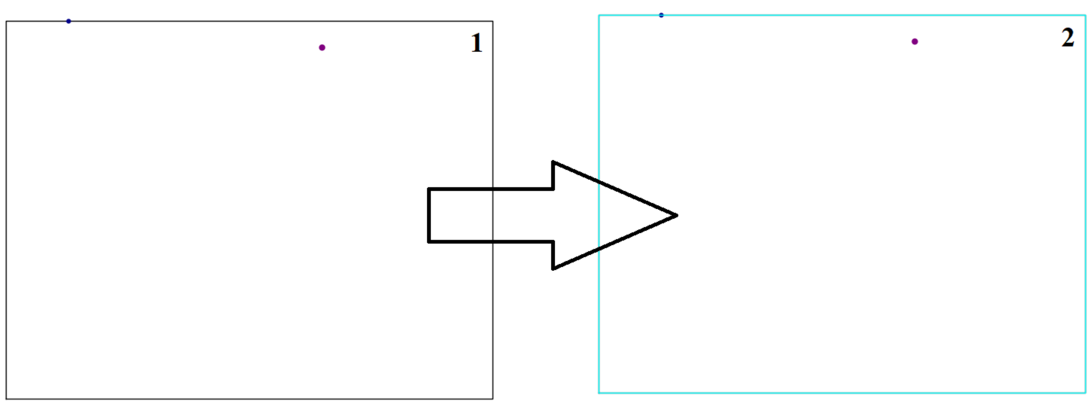
2.1.1 Budowanie drzewa

Korzeń drzewa symbolizuje największy prostokąt, zaznaczony kolorem czarnym. Aktulanie wstawiany punkt do drzewa jest pogrubiony i pokolorowany na fiolerowo. Jeśli można w danym węźle wstawić punkt, to jego kolor zmienia się na ciemnoniebieski i traci swoją początkową grubość.



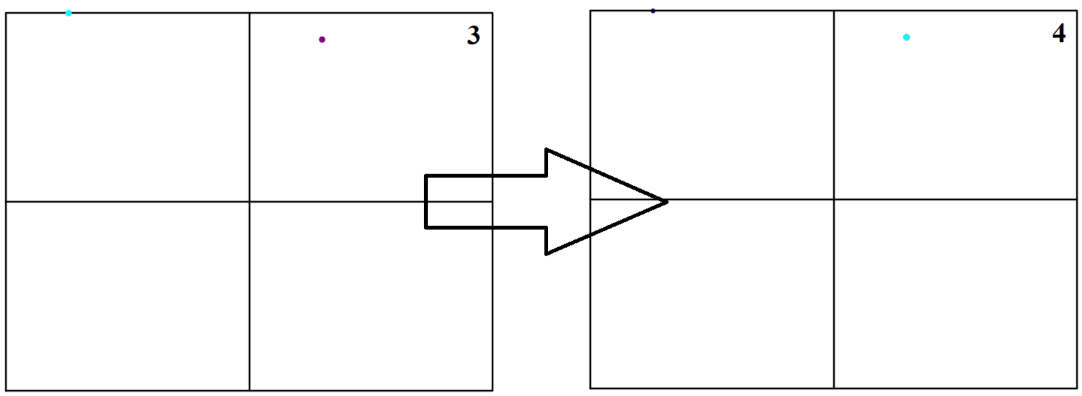
Wizualizacja 4: Budowanie drzewa: wstawienie punktu

Jeśli węzeł wymaga podziału, to jest on zaznaczany kolorem cyjankowym.



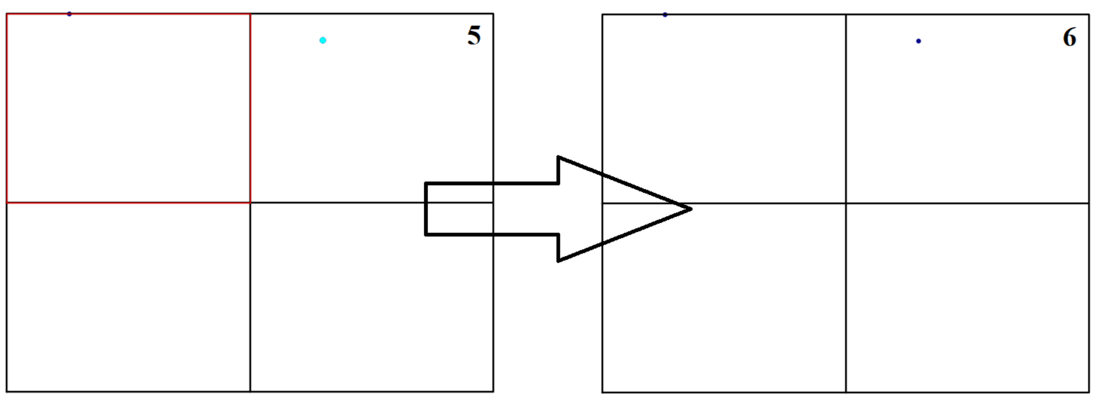
Wizualizacja 5: Budowanie drzewa: podział - rozdzielenie węzła

Następnie, kolorem czarnym, rysowane są linie podziału, a wszyskie punkty danego węzła są rozdysponowywane do odpowiednich dzieci. To jaki punkt jest aktulanie rozważany jest zaznaczone kolorem cyjankowym, dodatkowo dany punkt jest pogrubiony.



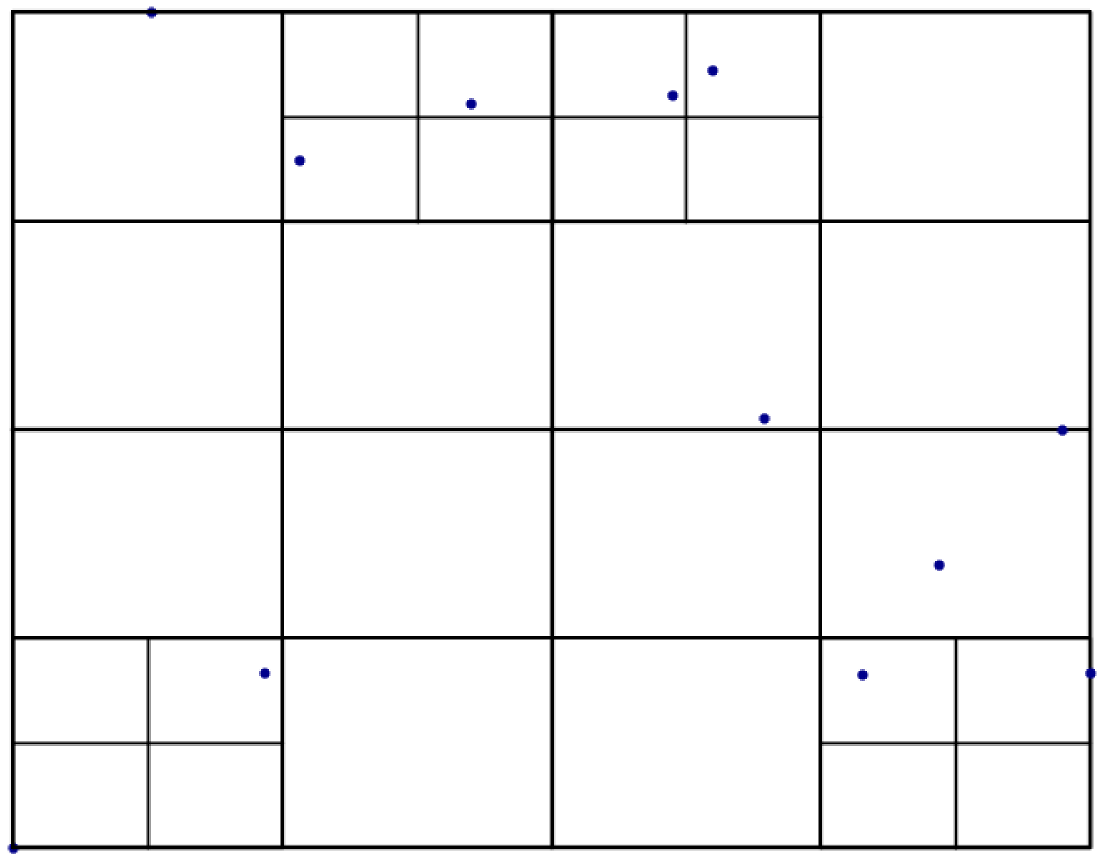
Wizualizacja 6: Budowanie drzewa: podział - przydział punktów do dzieci danego węzła

W przypadku kiedy nie można wstawić punktu w danym węźle, jest on kolorowany na czerwono.



Wizualizacja 7: Budowanie drzewa: brak możliwośći wstawienia punktu do węzła

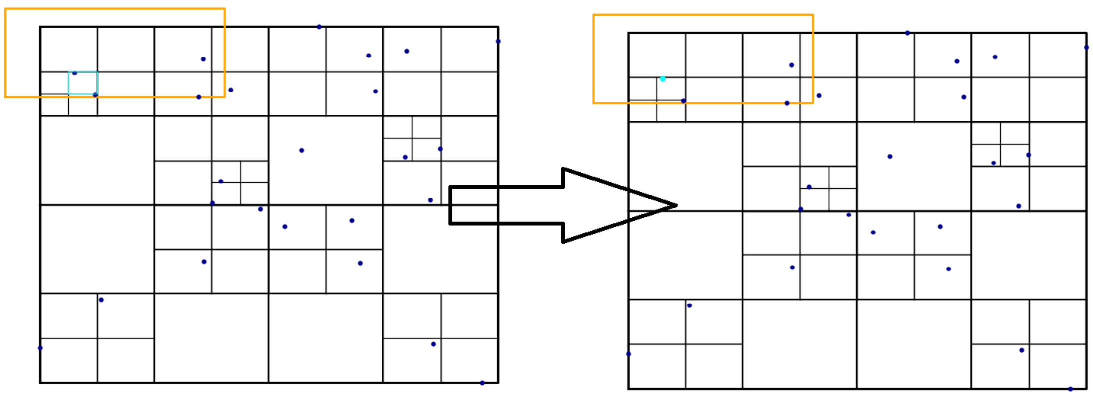
W pełni wybudowane drzewo prezentuje się w następujący sposób:



Wizualizacja 8: Budowanie drzewa: drzewo w pełni wybudowane

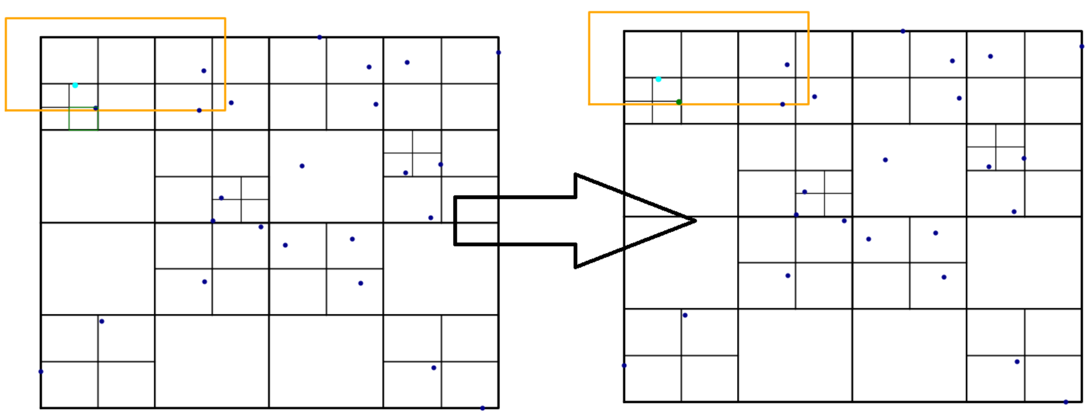
2.1.2 Wyszukiwanie w zadanym prostkącie

Prostokąt, w którym należy znaleźć punkty jest zaznaczony na pomarańczowo. Drzewo jest uprzednio wybudowane, aby móc dokładnie śledzić, które węzły są rozważane i jak wygląda ogólna struktura drzewa. W trakcie wizualizacji krokowej zaznaczane są 3 sytuacje:
Jeśli dany węzeł w pełni się zawiera w obszarze przeszukiwania, to jest on kolorowany na cyjankowo, a wszysktie punkty w nim zawarte, są dodawane do zbioru wynikowego. Dodane w ten sposób punkty są pogrubione i oznaczone kolorem cyjankowym.



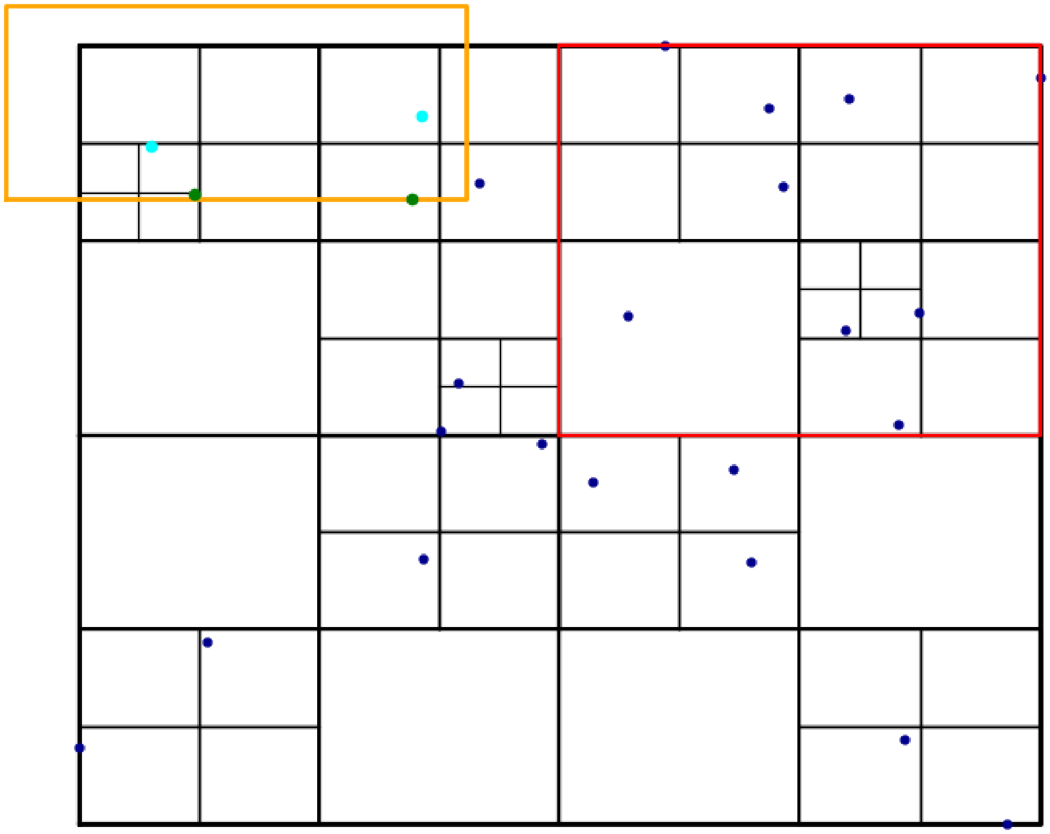
Wizualizacja 9: Przeszukiwanie obszaru: węzeł w pełni zawarty w zadanym obszarze

Jeśli dany węzeł przecina obszar przeszukiwania, to jest kolorowany na zielono, a wszystkie punkty, które należą do obszaru dodawane są do zbioru wynikowego. Dodane w ten sposób punkty są pogrubione i oznaczone kolorem zielonym.



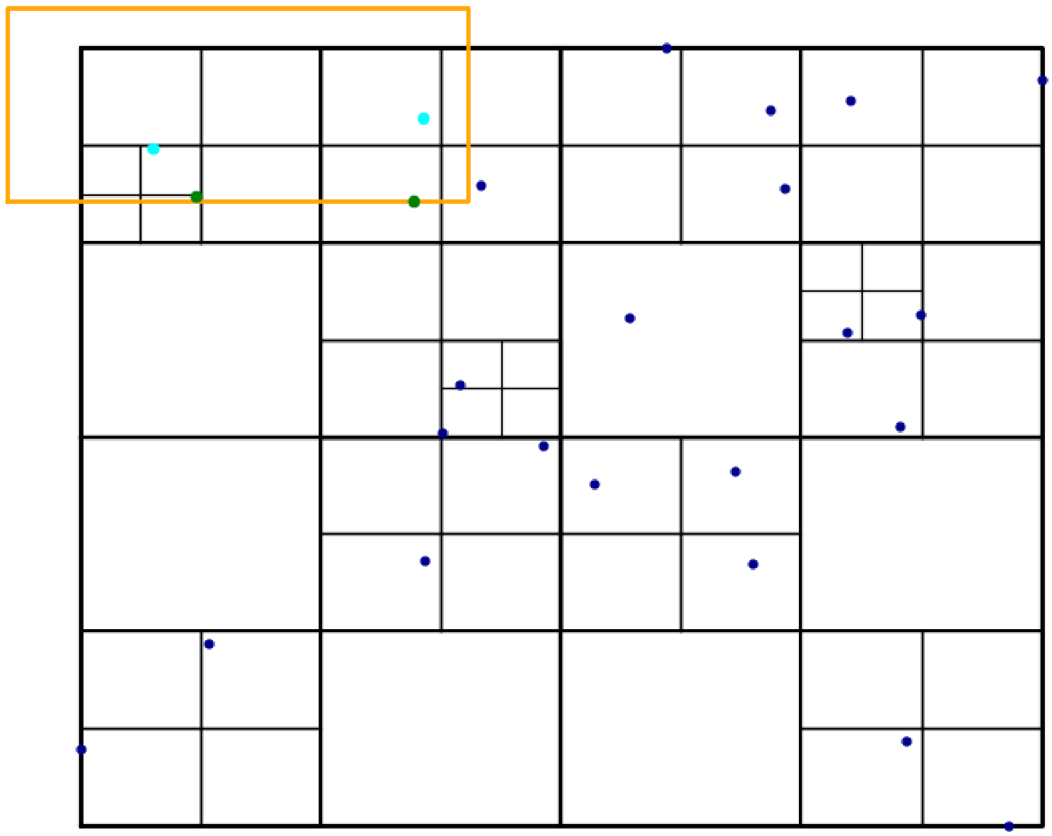
Wizualizacja 10: Przeszukiwanie obszaru: węzeł przecina zadany obszar

Jeśli dany węzeł nie przecina obszaru przeszukiwania, to jest on kolorowany na czerwono.



Wizualizacja 11: Przeszukiwanie obszaru: węzeł nie przecina zadanego obszaru

Po skończonym algorytmie wyszukiwania wizalizacja prezentuje się w następujący sposób:

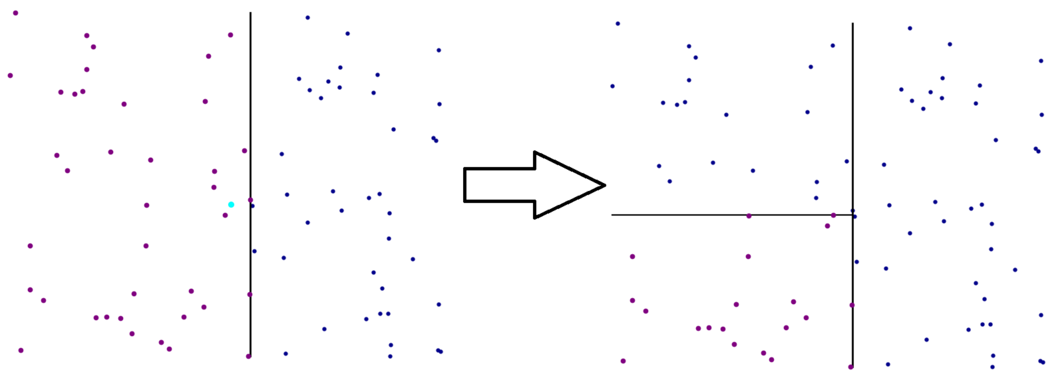


Wizualizacja 12: Przeszukiwanie obszaru: pełny zbiór wynikowy

2.2 KD-Tree

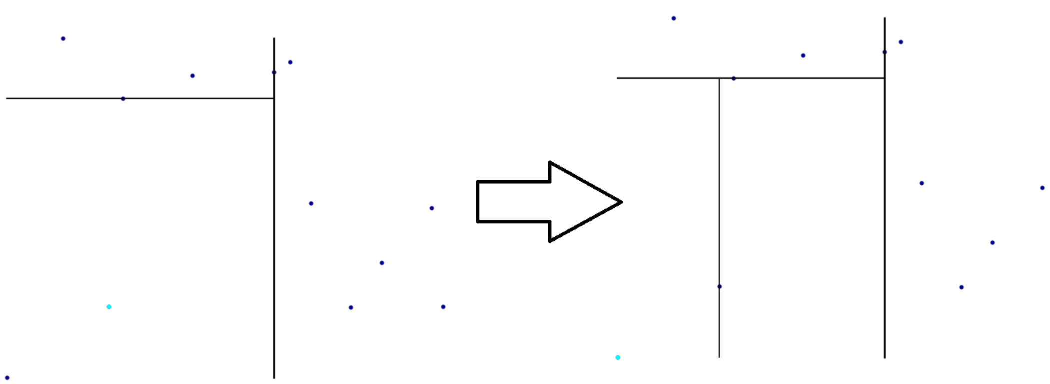
2.2.1 Budowanie drzewa

Wymiar rysowania gałęzi jest wybierany na przemian, zaczynając od pierwszego wymiaru. Wyznaczana jest mediana spośród punktów, a następnie rysowana jest gałąź (w kolorze czarnym). Aktualna mediana jest oznaczona kolorem cyjankowym. Rozważane punkty, z których została wybrana są oznaczone na fioletowo, reszta puntków jest oznaczona kolorem ciemnoniebieskim.



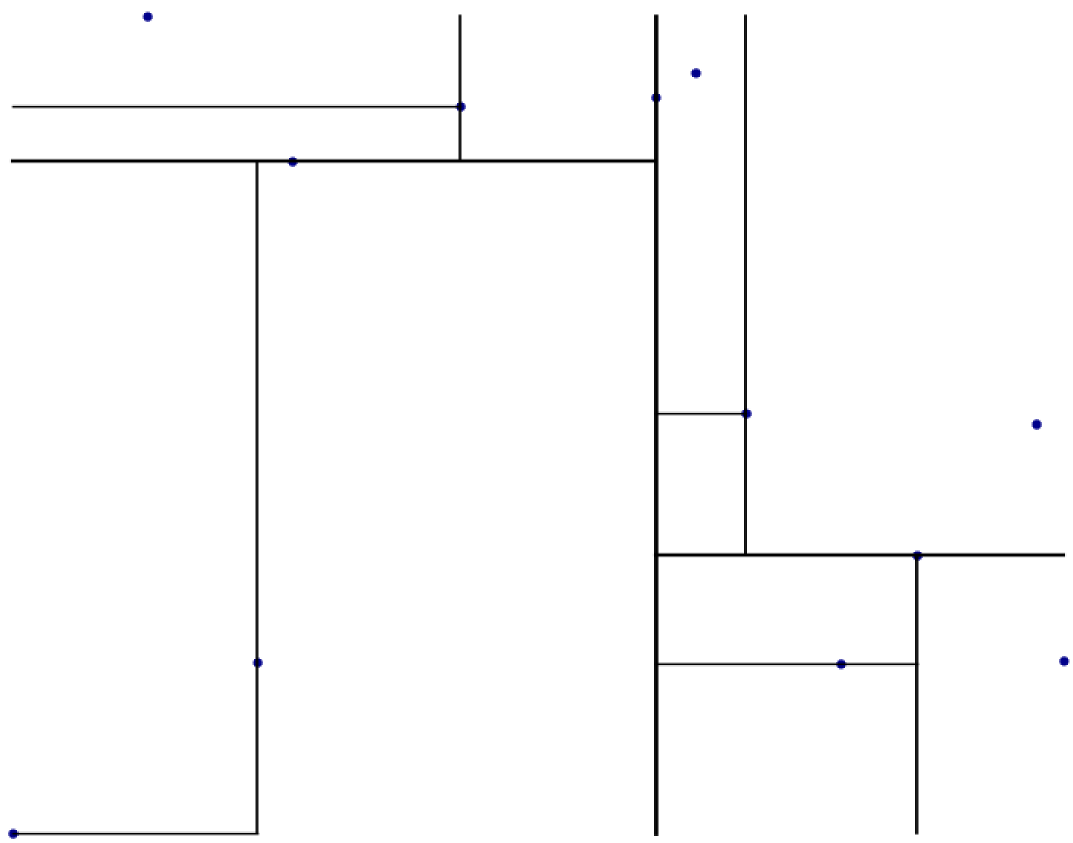
Wizualizacja 13: Budowanie drzewa

Budowanie trwa do momentu, aż wyliczana jest mediana dla więcej niż jednego punktu, wpp. pojedyńczy punkt jest liściem i nie jest łączony na wizualizacji.



Wizualizacja 14: Budowanie drzewa cd.

Po zakończonym budowaniu wizualizacja wygląda w następujący sposób:

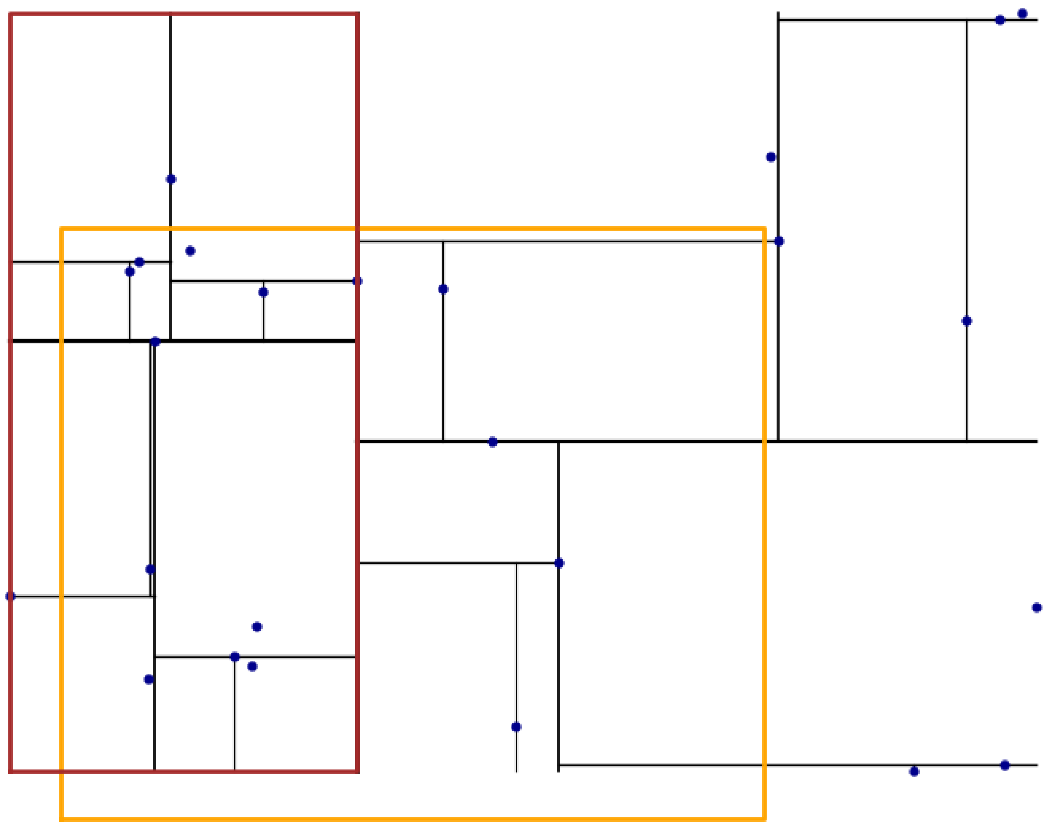


Wizualizacja 15: Budowanie drzewa: drzewo w pełni wybudowanie

2.2.2 Wyszukiwanie w zadanym prostkącie

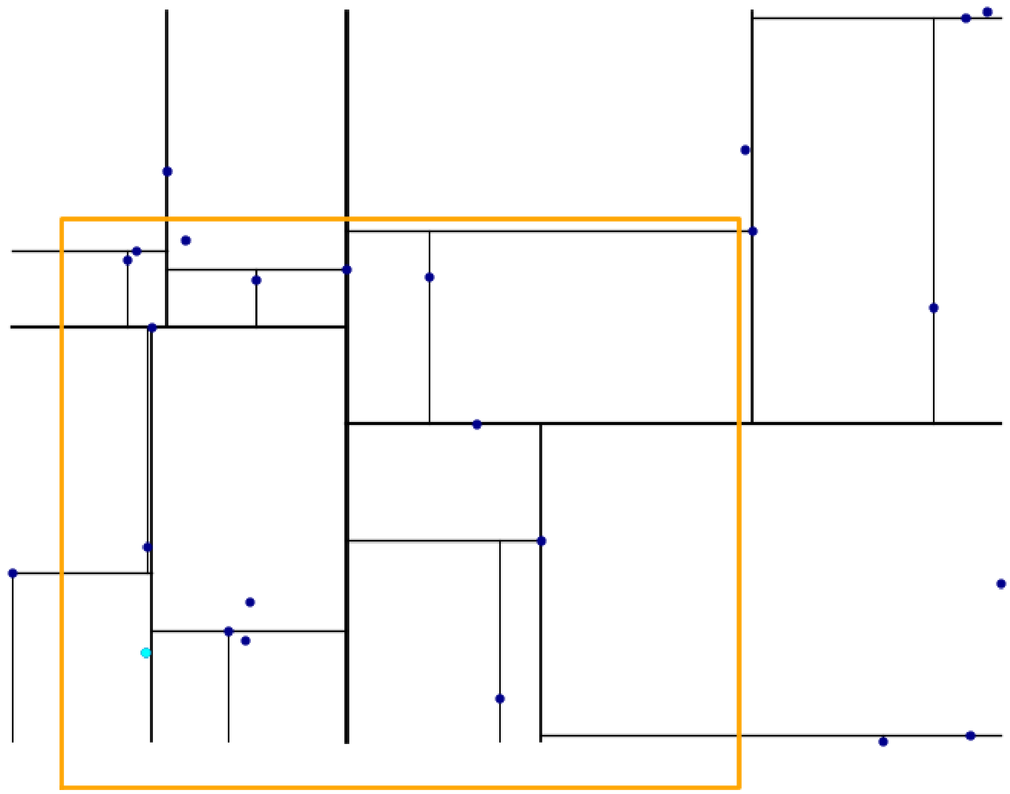
Podobnie jak w przypadku **Quad-Tree** wizualizowane jest w pełni wybudowane drzewo, a obszar przeszukiwania jest oznaczony kolorem pomarańczowym. Możemy wyróżnić 3 sytuacje, które są wizualizowane:

Aktualnie rozważany obszar jest oznaczany kolorem brązowym



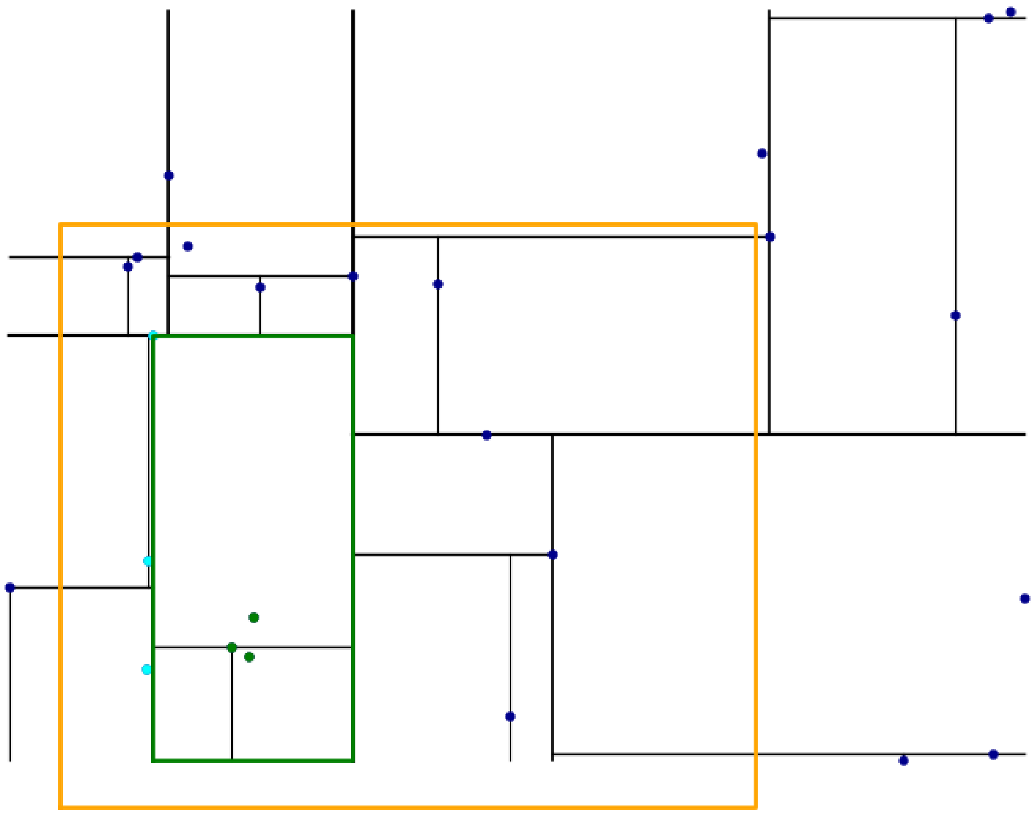
Wizualizacja 16: Przeszukiwanie obszaru: rozważany obszar

Jeśli rozważany obszar zawierał 2 punkty, to następnie każdy z nich jest rozważany względem należenia do obszaru przeszukiwania. Jeśli dany punkt należy do obszaru, to jest to zaznaczone poprzez pokolorowanie go kolorem cyjankowym.



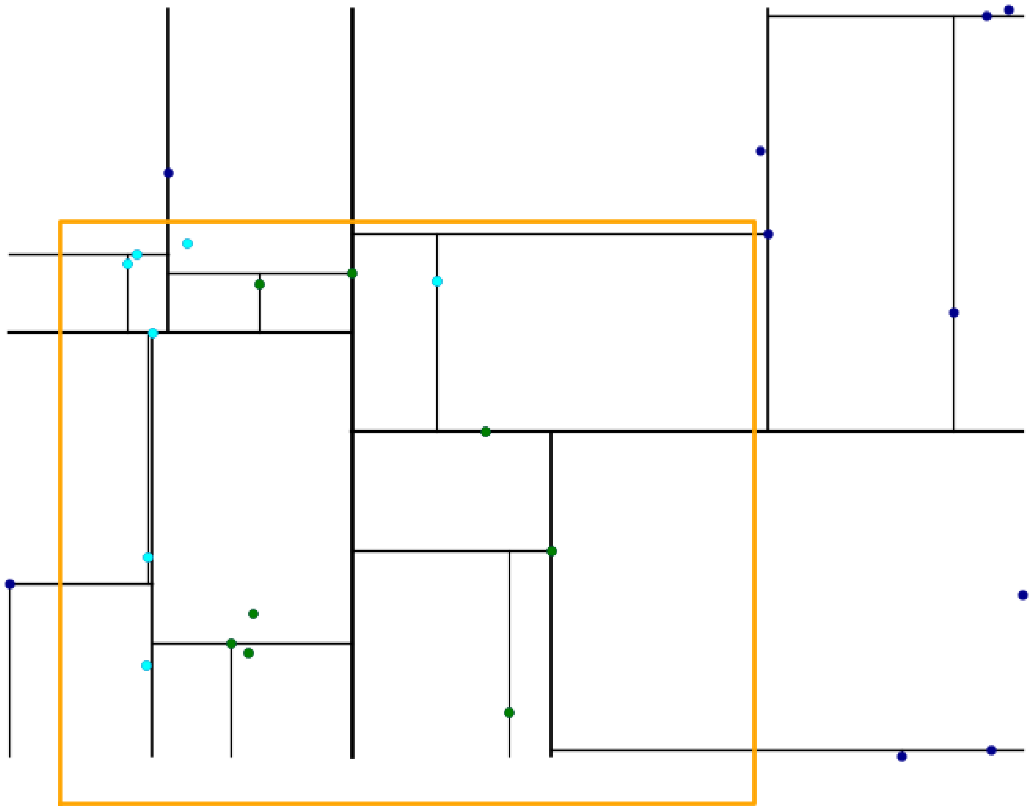
Wizualizacja 17: Przeszukiwanie obszaru: liść należący do zbioru wynikowego

Jeśli rozważany obszar w całości zawiera się w obszarze przeszukiwania, to zaznaczny jest kolorem zielonym, a wszyskie jego punkty zostają dodane do zbioru wynikowego i również oznaczane są na zielono.



Wizualizacja 18: Przeszukiwanie obszaru: prostokąt zawarty w obszarze przeszukiwania

Po skończonym algorytmie wyszukiwania wizalizacja prezentuje się w następujący sposób:



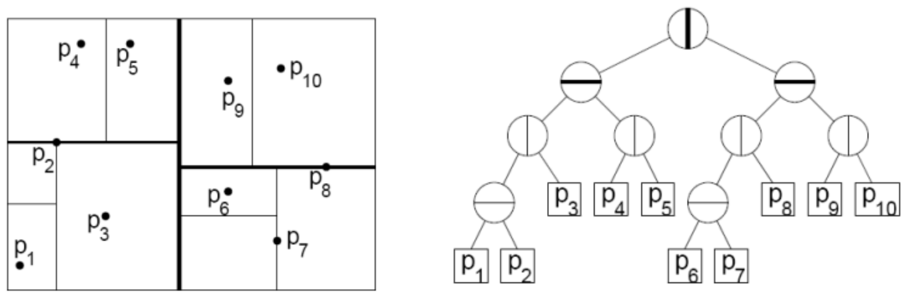
Wizualizacja 19: Przeszukiwanie obszaru: pełny zbiór wynikowy

Część IV
Sprawozdanie

1 Wstęp teoretyczny

1.1 Struktura KD-Tree

KD-Tree (k-dimensional) jest drzewem binarnym używanym do podziału punktów określonych w k-wymiarowej przestrzeni. Struktura tworzona jest w następujący sposób: spośród zadanych punktów wybieramy medianę przy pomocy funkcji **QuickSelect**. Należy przed tym wybrać wymiar przestrzeni względem, którego porównywane będą punkty. W tej implementacji każdy kolejny podział jest dokonywany względem wymiaru o jeden większego niż poprzedni. Podział punktów na dwa równe podzbiory (punktów mniejszych oraz punktów większych od mediany) odpowiada utworzeniu dwóch nowych gałęzi wychodzących z węzła głównego. Węzły przechowują informacje, względem którego wymiaru oraz jakiej wartości nastąpił podział, jaki jest minimalny przedział wielowymiarowy, który reprezentuje węzeł oraz jakie punkty należą do tego obszaru. Podziału przestrzeni dokonujemy, jeżeli obszar do podziału zawiera więcej niż jeden punkt. Jeżeli zawiera dokładnie jeden, ten jest umieszczany w drzewie jako liść wychodzący z węzła reprezentującego tę podprzestrzeń do jakiej punkt należy. Ten schemat postępowania, dla przykładowych danych w dwuwymiarowej przestrzeni ilustruje poniższy rysunek.

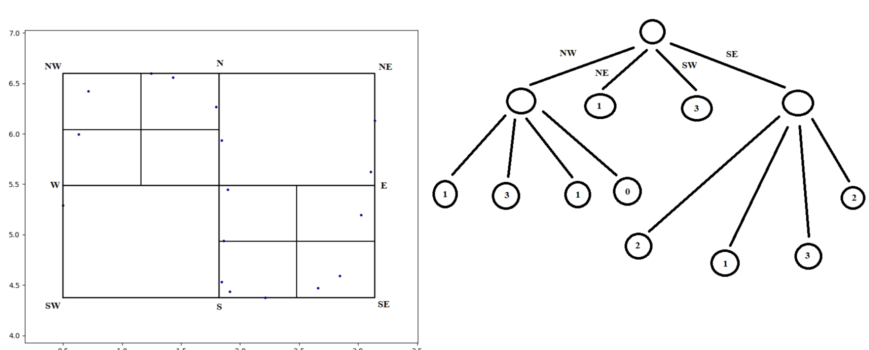


Wizualizacja 20: Schemat struktury KD-Tree

Algorytm przeszukiwania punktów również jest rekurencyjną funkcją realizującą następujący schemat dla zadanego węzła.
1. Jeżeli węzeł jest liściem (punktem) dodajemy do zbioru wynikowego.
2. Jeżeli węzeł reprezentuje prostokąt, który w całości zawiera się w zadanym, do zbioru wynikowego dodajemy wszystkie punkty należące do danego węzła.
3. Jeżeli węzeł reprezentuje prostokąt, który przecina się z tym zadanym, wywołujemy funkcje dla obu gałęzi węzła.
Złożoność budowania drzewa *O(nlogn)* wynika z tego, że na każdym poziomie drzewa należy wywołać funkcję QuickSelect o złożoności *O(n)*. Samych poziomów jest *logn*, ponieważ jest to zbalansowane drzewo BST. Dzięki prędkiemu dobieraniu wymiarów, względem, których dzielona jest przestrzeń, złożoność budowania drzewa będzie równa dokładnie *O(nlogn)*. Inne implementacje mają gorszą złożoność zależną od liczby wymiarów O(knlogn), ale za to pozwalają uzyskać nieco lepszą strukturę drzewa.
Złożoność wyszukiwania oraz zliczania punktów w zadanym obszarze wynosi $O(\sqrt{n})$. Dzieje się tak dzięki temu, że każdy węzeł drzewa przechowuje listę punktów należących do obszaru reprezentowanego przez niego. Zatem kosztem pamięci uzyskuje się lepszy czas.

1.2 Struktura Quad-Tree

Quad-Tree jest drzewem ćwiartkowym. Wykorzystuje się tę strukturę w celu podziału przestrzeni dwuwymiarowej. Podziały są tworzone poprzez dzielenie rozważanego obszaru na cztery równe części, a jeśli jest taka potrzeba, to podział jest wykonywany ponownie. W opracowaniu tego projektu przyjęliśmy, że podział nastąpi w momencie, gdy w danym prostokącie będzie znajdować się więcej niż 3 punkty. W tym wypadku na każde pole maksymalnie przypadają 3 punkty. W celu podziału przestrzeni, można też zadać określoną głębokość drzewa tzn. ile maksymalnie ma zostać wykonanych podziałów, jednak taka alternatywa nie została przez nas przyjęta.



Wizualizacja 21: Schemat struktury Quad-Tree

Na powyższej wizualizacji widać schemat budowy drzewa ćwiartkowego, prostokąty powstałe w wyniku podziału są oznaczone: **NW** - lewy górny, **NE** - prawy górny, **SW** - lewy dolny, **SE** - prawy dolny. Po lewej stronie widać utworzone podziały na płaszczyźnie, a po prawej strukturę drzewa i połączenia między węzłami. Dla łatwiejszego odczytu wizualizacji w liściach podano liczbę przechowywanych punktów.

Algorytm przeszukiwania jest wykonywany w następujący sposób:
Przeszukiwanie zaczynamy od korzenia, dla każdego odwiedzonego węzła są trzy możliwości

1. jeśli prostokąt odwiedzonego węzła nie przecina się z zadanym obszarem przeszukiwania, to żaden punkt przypisany temu tego węzłowi nie należy do interesującego nas obszaru - zaprzestajemy przeszukiwać gałąź

2. jeśli prostokąt odwiedzonego węzła zawiera się całkowicie w zadanym obszarze przeszukiwania, to wszystkie jego punkty dodajemy do zbioru wynikowego - bierzemy całą gałąź

3. jeśli prostokąt odwiedzonego węzła częściowo zawiera się w zadnym obszarze przeszukiwania, to:

   - jeśli węzeł jest liściem, to do zbioru wynikowego dodajemy te punkty, które należą do zadanego obszaru przeszukiwania
   - wpp. odwiedzamy jego dzieci - wchodzimy wgłąb drzewa

Złożoność czasowa oraz pamięciowa budowania struktury **Quad-Tree** jest zależna od głębokości drzewa. Sama głębokość drzewa zależy od tego jak punkty danego zbioru są rozmieszczone na płaszczyźnie. Wynosi ona $(n(d+1))$, gdzie n jest liczebnością zbioru punktów, a d (ang. *depth*), to głębokość drzewa. Innymi słowy, dla każdego punktu w pesymistycznym przypadku jestśmy zmuszeni przejść na najniższy poziom głębokośći, dokonać podziału i wstawić punkt. Na głębokość wpływa też pojemność liści tzw. **capacity** - im większa pojemność tym, drzewo będzie płytsze, bo zmniejszy się liczba potrzebnych podziałów. W przypadku gdzie **capacity** byłoby równe liczbie liści, złożoność czasowa i pamięciowa budowania wyniesie $O(n)$, ale takie rozwiązanie jest tożsame z rozwiązaniem trywialnym w kontekście wyszukiwania punktów, bo w tym wypadku zawsze sprawdzane są wszyskie punkty, co jest nieefektywne. (Jest to szerzej omówione w sekcji **2.7**)

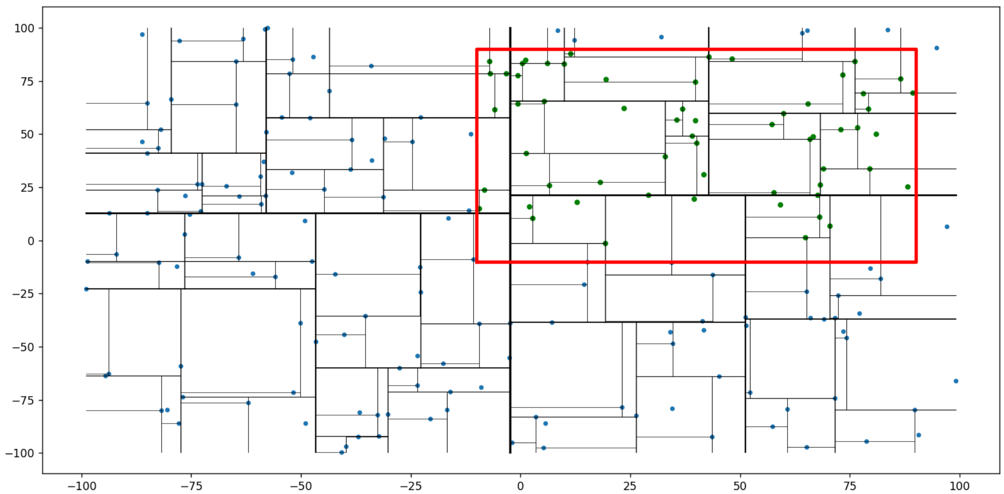
Złożoność czasowa wyszukiwania punktów w zadanym obszarze z wykorzystaniem struktury **QuadTree** jest również zależna od głębokośći drzewa d, ale też liczby liści, które częściowo zawierają się w przeszukiwanym obszarze u nas oznaczone jako l (ang. *leaf*). Jest to uzasadniodnie tym, że dla każdego dokonanego podziału, należy sprawdzić każdy przecinający się prostokąt z zadanym obszarem przeszukiwań, podczas gdy inne przypadki (wyżej wymienione jako 1 i 2) nie tworzą kolejnych rozgałęzień w drzewie przeszukiwania. Zatem złożoność czasowa tej operacji wynosi $O(ld)$.

2 Testowanie dla różnych danych wejściowych

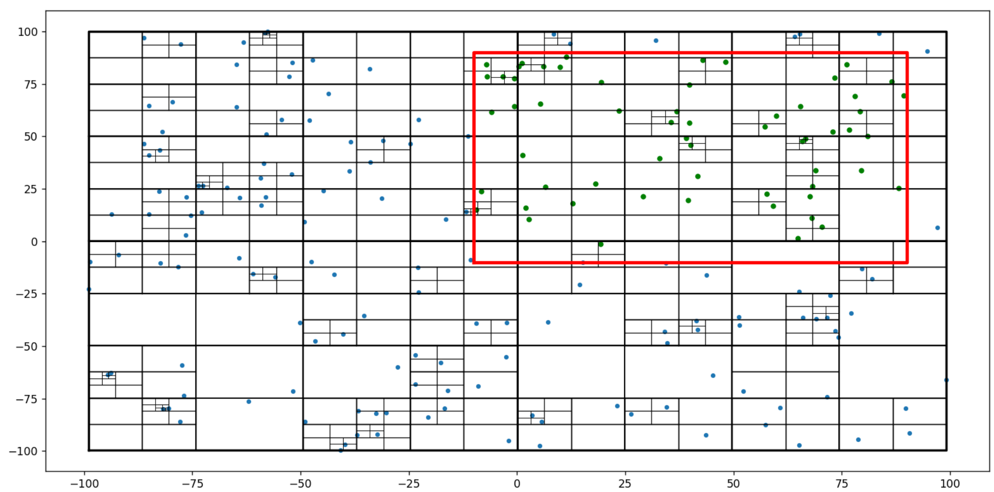
Na wstępie warto dodać, iż podczas analizy czasowej drzew wartość **capacity** w **Quad-Tree** będzie ustawiona na 3. Dzięki temu tworzenie drzewa będzie trwało o wiele krócej, a stała podczas przeszukiwania nadal nie będzie duża. W przypadku analizy poprawności znajdowanych zbiorów punktowych, dla lepszej widoczności wizualizacji, **capacity** w **Quad-Tree** będzie ustawione na 1.

2.1 Zbiór punktów o rozkładzie jednostajnym

Jest to zbiór losowo wybranych punktów należących do prostokąta. W związku z jednostajnością rozkładu, punkty są umieszczone równomiernie po całym obszarze. Prostokąt, w którym szukane są punkty stanowi ćwierć całego obszaru oraz w całości się w nim zawiera. Poniżej znajduje się wizualizacja dla niewielkich danych - 200 punktów o współrzędnych z przedziału $[-100, 100]$.



Wizualizacja 22: Wynik dla rozkładu jednostajnego, **KD-Tree**



Wizualizacja 23: Wynik dla rozkładu jednostajnego, **Quad-Tree**

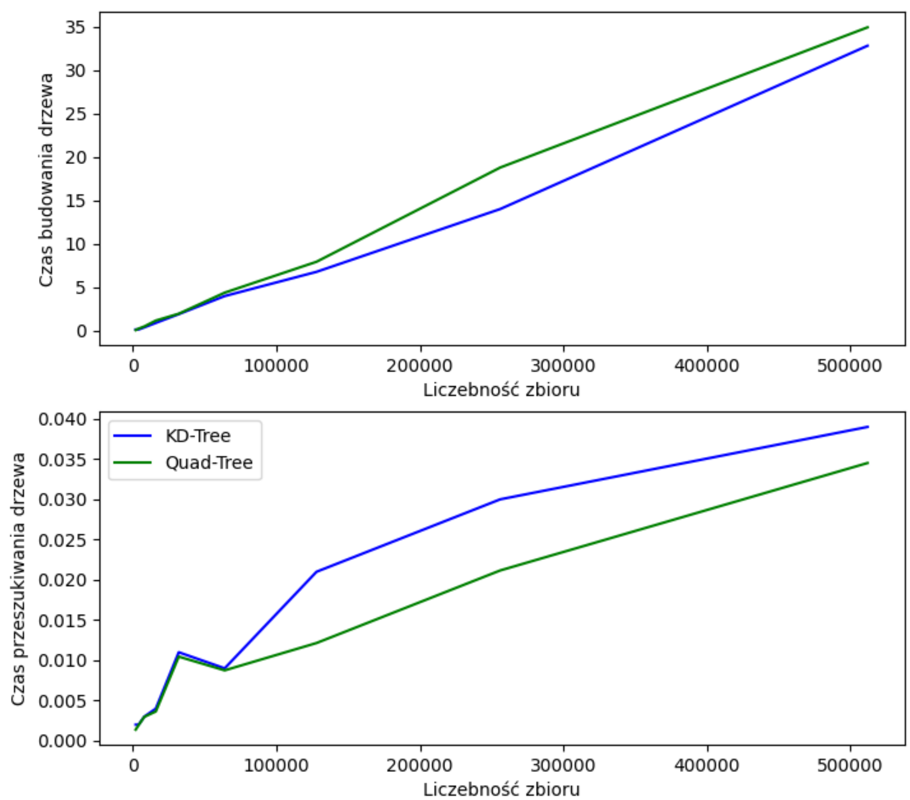
Jak widać na **Wizualizacjach 22 i 23**, algorytmy poprawnie znalazły punkty w czerwonym prostokącie. Warto dodać, że oba algorytmy znalazły dokładnie 59 punktów.
Poniżej zamieszczona jest tabela z czasami algorytmów (budowanie drzew oraz przeszukiwanie geometryczne) dla rozkładu jednostajnego o różnej liczebności. Współrzędne punktów losowane są z przedziału $[-1000, 1000]$, a szukane punkty spełniają następujący warunek:

$$-100 \leq x \leq 900, -100 \leq y \leq 900$$

Dane w tabeli są zaokrąglane do 3 miejsc po przecinku.

Liczebność zbioru	Liczba znalezionych punktów	Czas tworzenia KD-Tree [s]	Czas tworzenia Quad-Tree [s]	Czas przeszukiwania KDTree [s]	Czas przeszukiwania QuadTree [s]
2000	531	0.131	0.068	0.002	0.001
4000	940	0.131	0.232	0.002	0.002
8000	1969	0.377	0.484	0.003	0.003
16 000	4071	0.89	1.159	0.004	0.004
32 000	8059	1.901	1.95	0.011	0.01
64 000	15840	3.984	4.378	0.009	0.009
128 000	32117	6.778	7.939	0.021	0.012
256 000	63846	14.016	18.799	0.03	0.021
512 000	127854	32.84	34.96	0.039	0.035

Tabela 1: Wyniki pomiarów czasu dla zbioru o rozkładzie jednostajnym

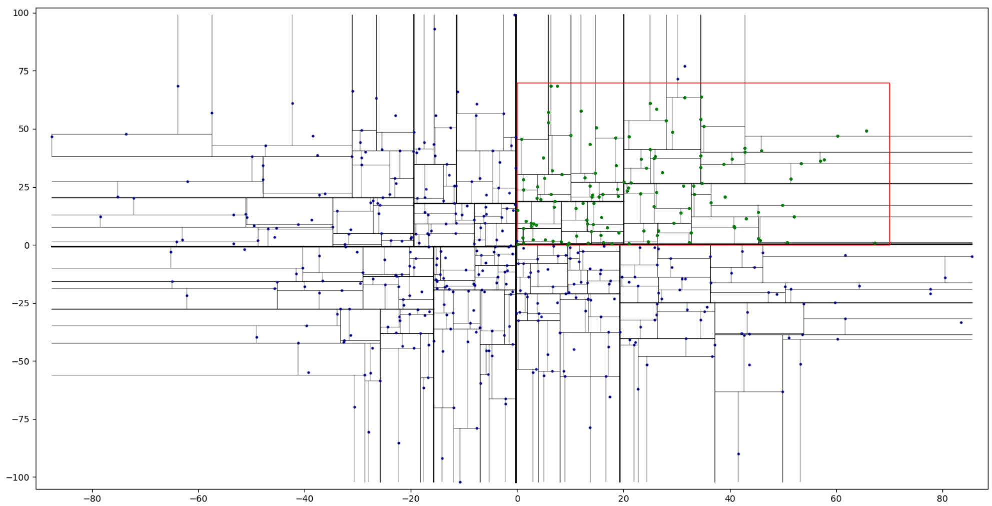


Wizualizacja 24: Wyniki pomiarów czasu dla zbioru o rozkładzie jednostajnym

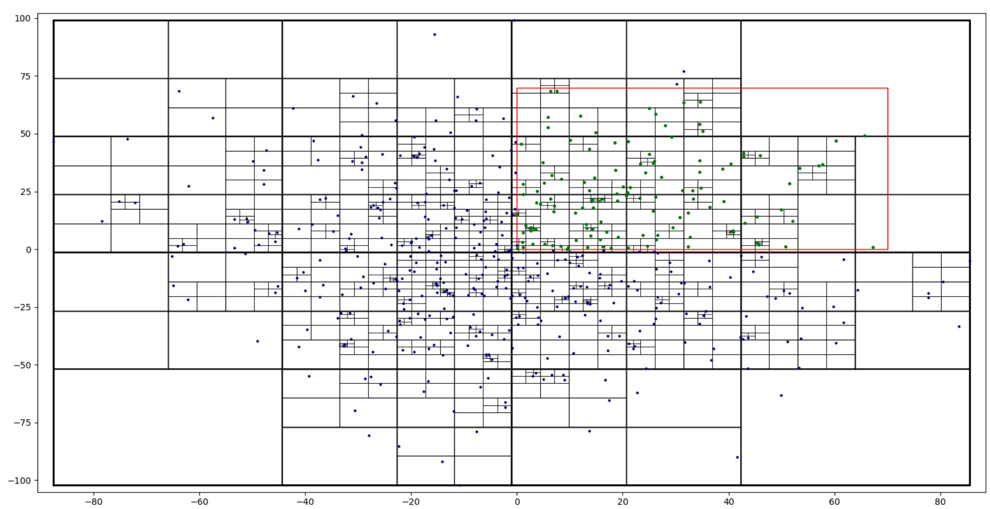
Jak widać w **Tabeli 1** oraz na **Wizualizacji 24**, budowanie drzewa zachodzi szybciej w przypadku **KD-Tree**. **Quad-Tree** buduje się średnio o 18% dłużej, natomiast wyszukiwanie trwa średnio o 11% szybciej.

2.2 Zbiów punktów o rozkładzie normalnym

Jest to zbiór punktów o współrzędnych wylosowanych zgodnie z rozkładem Gaussa. Prostokąt, w którym szukane są punkty stanowi, tak jak w poprzednim przypadku, około ćwierć obszaru z punktami. Poniżej znajduje się wizualizacja dla przykładowego zbioru 500 punktów.



Wizualizacja 25: Wynik dla rozkładu normalnego, **KD-Tree**



Wizualizacja 26: Wynik dla rozkładu normalnego, **Quad-Tree**

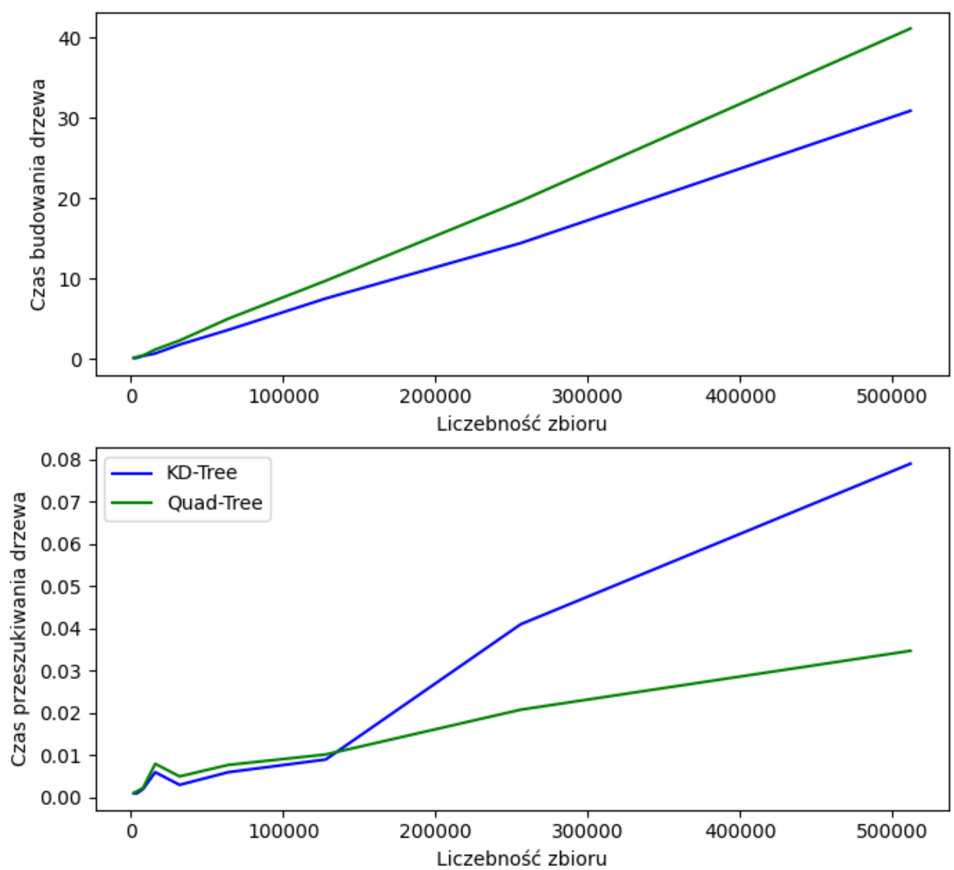
Jak widać na **Wizualizacjach 25 i 26**, oba drzewa umożliwiły w sposób poprawny przeszukanie obszaru. Poniżej zamieszczona jest tabela z czasami algorytmów dla zbiorów o rozkładzie normalnym o różnej liczebności. Szukane punkty spełniają następujący warunek:

$$-100 \leq x \leq 900, -100 \leq y \leq 900$$

Dane w tabeli są zaokrąglane do 3 miejsc po przecinku.

Liczebność zbioru	Liczba znalezionych punktów	Czas tworzenia KD-Tree [s]	Czas tworzenia Quad-Tree [s]	Czas przeszukiwania KDTree [s]	Czas przeszukiwania QuadTree [s]
2000	949	0.127	0.091	0.001	0.001
4000	1952	0.128	0.242	0.001	0.001
8000	3820	0.373	0.428	0.002	0.002
16 000	7633	0.675	1.168	0.006	0.008
32 000	15321	1.779	2.283	0.003	0.005
64 000	30699	3.607	5.0	0.006	0.008
128 000	61142	7.517	9.721	0.009	0.01
256 000	122358	14.403	19.635	0.041	0.021
512 000	244705	30.862	41.093	0.079	0.035

Tabela 2: Wyniki pomiarów czasu dla zbioru o rozkładzie normalym

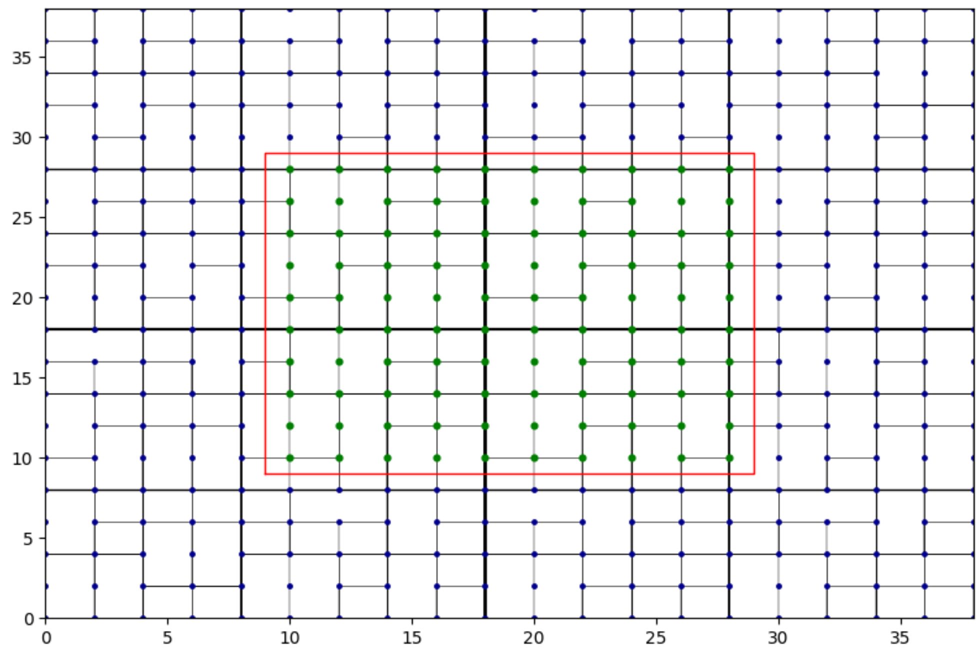


Wizualizacja 27: Wyniki pomiarów czasu dla zbioru o rozkładzie Gaussa

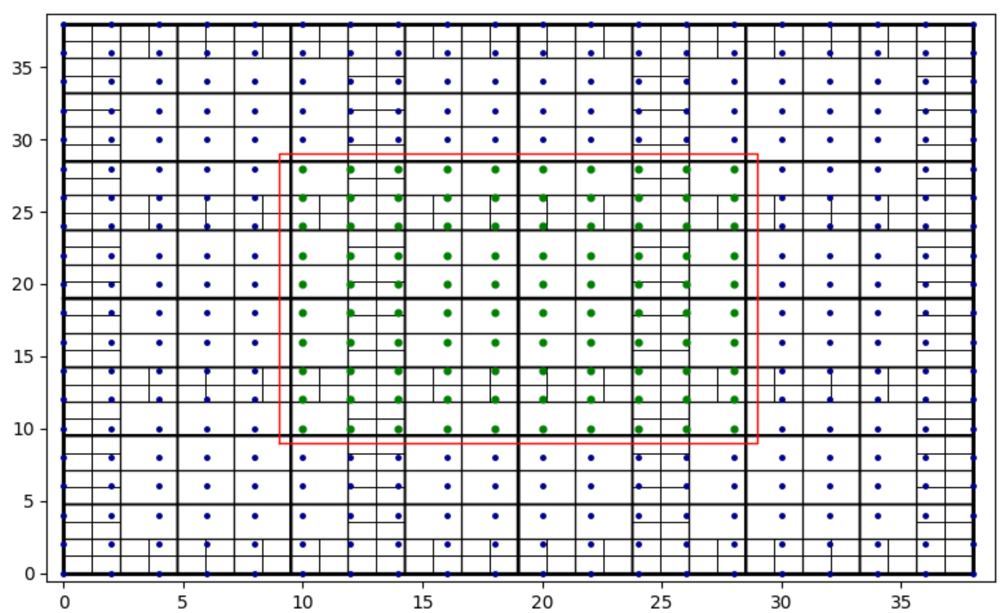
Podobnie jak w przypadku rozkładu jednostajnego, wykorzystywanie **KD-Tree** jest korzystniejsze w przypadku budowania. Według danych z **Tabeli 2** i **Wizualizacji 27**, tym razem budowanie drzewa ćwiartkowego zajmuje średnio o 35% więcej czasu, a przeszukiwanie trwa około 14% dłużej (obliczane na podstawie niezaokrąglanych wartości).

2.3 Zbiór punktów o rozkładzie jednostajnym (siatka)

Jest to zbiór punktów, w którym punkty są umieszczone w rzędach w równej odległości od siebie. Poniżej zamieszczono przykładową wizualizację wyników dla takiej siatki 20x20 punktów odległych od siebie o 2 jednostki.



Wizualizacja 28: Wynik dla siatki, **KD-Tree**



Wizualizacja 29: Wynik dla siatki, **Quad-Tree**

W obu przypadkach zwróconych zostało 100 punktów, co pokrywa się z Wizualizacjami 28 i 29 oraz oczekiwaniami.

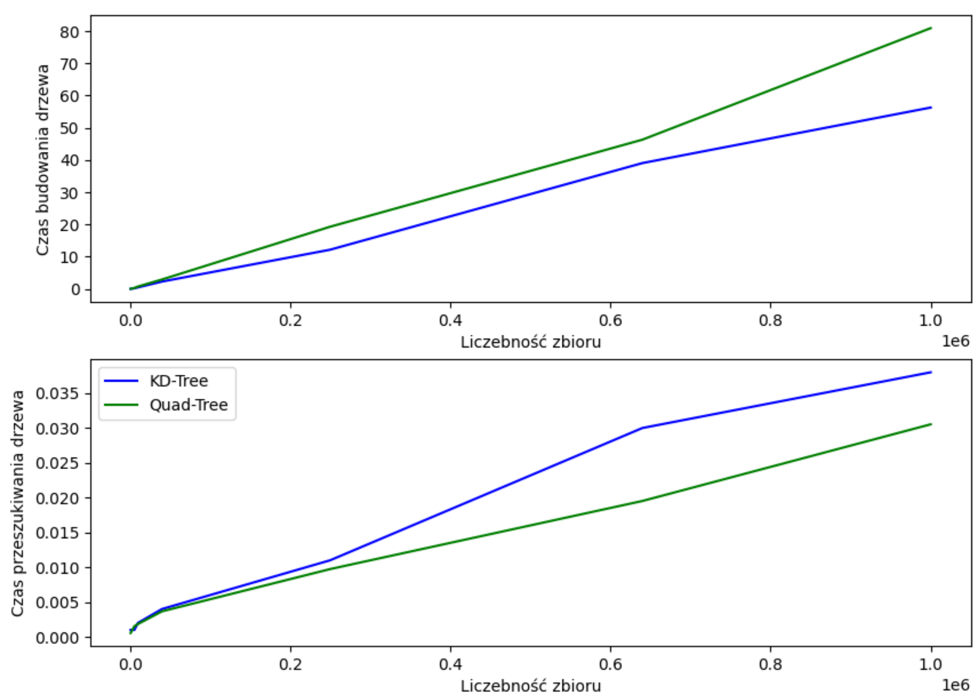
Poniżej zamieszczona jest tabela z czasami algorytmów dla siatek punktów o różnej liczebności. Szukane punkty spełniają następujący warunek:

$$0 \leq x \leq a/2, 0 \leq y \leq a/2, \text{ gdzie a jest liczbą punktów na boku siatki.}$$

Dane w tabeli są zaokrąglane do 3 miejsc po przecinku.

Liczebność zbioru	Liczba znalezionych punktów	Czas tworzenia KD-Tree [s]	Czas tworzenia Quad-Tree [s]	Czas przeszukiwania KDTree [s]	Czas przeszukiwania QuadTree [s]
30^2	256	0.031	0.049	0.001	0.001
75^2	1444	0.209	0.316	0.001	0.002
100^2	2601	0.478	0.752	0.002	0.002
200^2	10201	2.29	2.934	0.004	0.004
500^2	63001	12.177	19.304	0.011	0.01
800^2	160801	39.049	46.312	0.03	0.02
1000^2	251001	56.279	80.91	0.038	0.031

Tabela 3: Wyniki pomiarów czasu dla siatek punktów

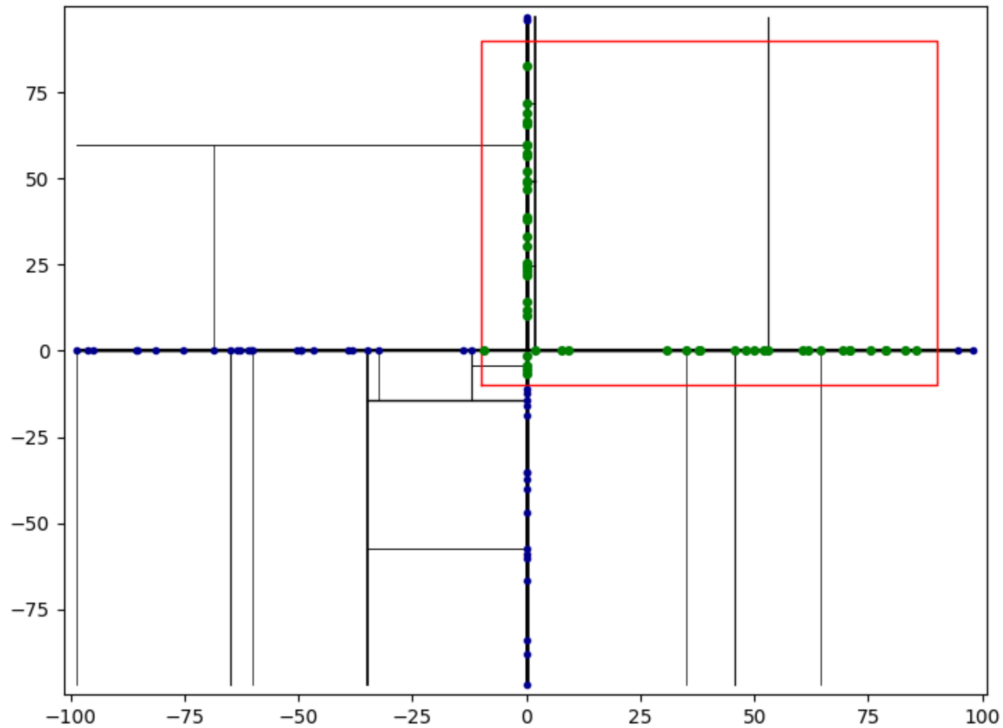


Wizualizacja 30: Wyniki pomiarów czasu dla siatek punktów

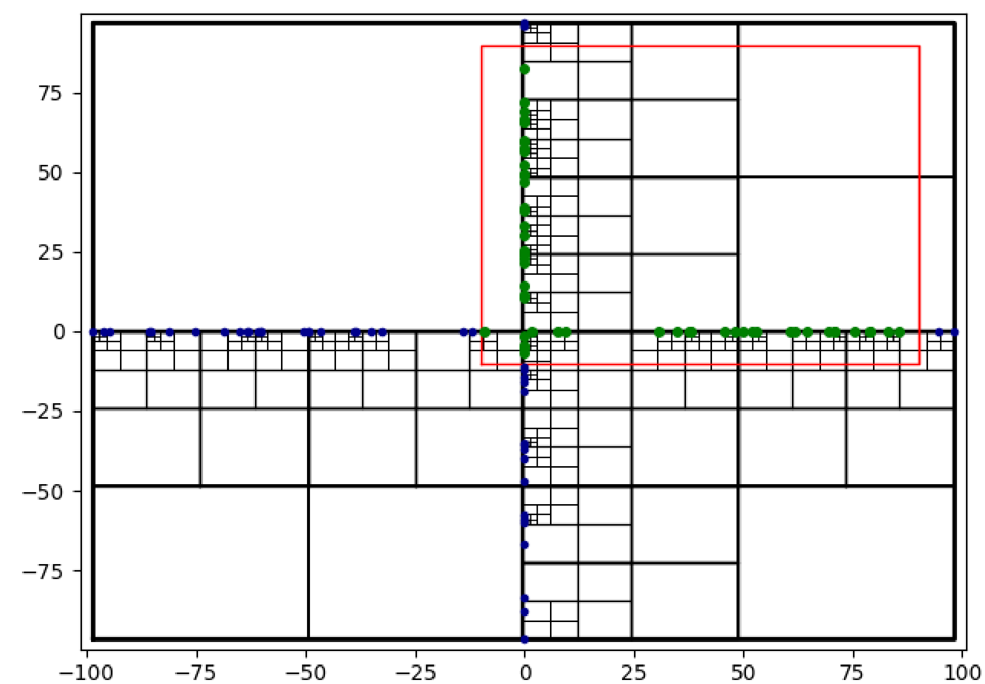
Według danych z **Tabeli 3** oraz **Wizualizacji 30** budowanie drzewa ćwiartkowego zajmuje średnio o 45% więcej niż budowanie drzewa k-wymiarowego. Przeszukiwanie natomiast zajmuje **Quad-Tree** średnio o 8% mniej czasu

2.4 Zbiór punktów w kształcie krzyża

Jest to zbiór punktów rozmieszczonych na osiach układu współrzędnego w taki sposób, że ich pierwsza połowa znajduje się na osi x, a druga połowa znajduje się na osi y. Poniżej zamieszczono przykładową wizualizację wyników algorytmów dla 100 punktów.



Wizualizacja 31: Wynik dla krzyża, **KD-Tree**



Wizualizacja 32: Wynik dla krzyża, **Quad-Tree**

Jak widać na **Wizualizacjach 31 i 32**, oba drzewa poprawnie zlokalizowały punkty.
Poniżej zamieszczona jest tabela z czasami algorytmów dla krzyża o różnej liczebności. Szukane punkty

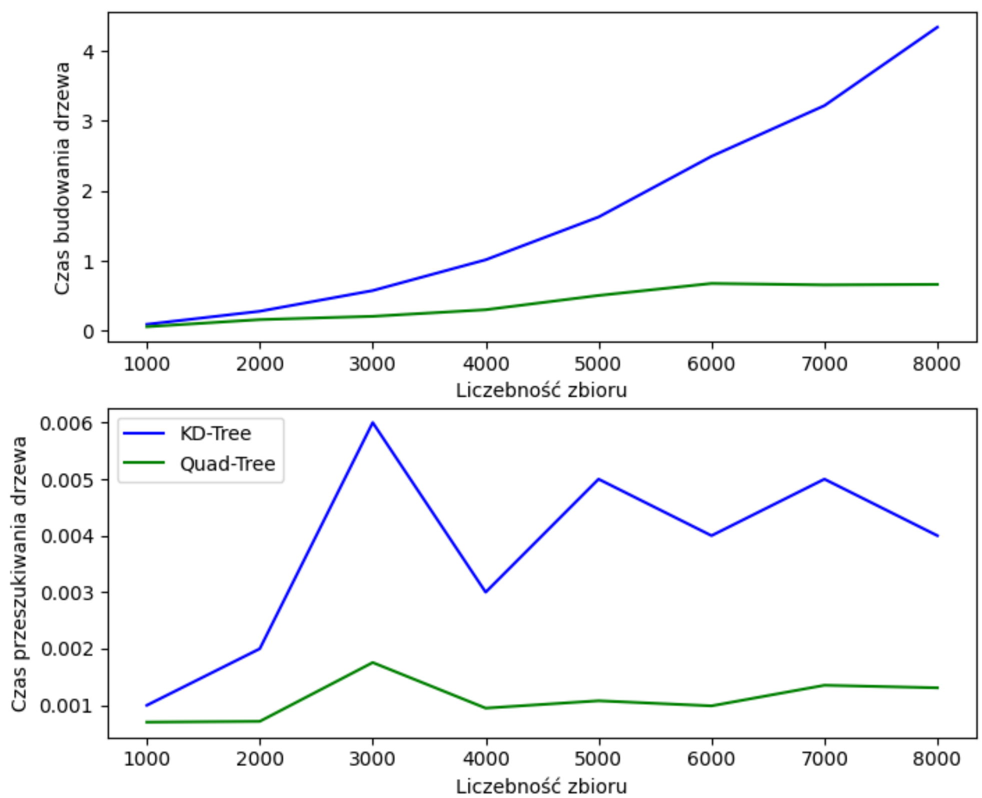
spełniają następujący warunek:

$$-100 \leq x \leq a - 100, -100 \leq y \leq a - 100,$$
gdzie a jest największą możliwą wartością, jaką współrzędna może przyjąć.

Dane w tabeli są zaokrąglane do 3 miejsc po przecinku.

Liczebność zbioru	Liczba znalezionych punktów	Czas tworzenia KD-Tree [s]	Czas tworzenia Quad-Tree [s]	Czas przeszukiwania KDTree [s]	Czas przeszukiwania QuadTree [s]
1000	508	0.092	0.058	0.001	0.001
2000	1006	0.279	0.16	0.002	0.001
3000	1480	0.575	0.207	0.006	0.002
4000	1952	1.015	0.301	0.003	0.001

Tabela 4: Wyniki pomiarów czasu dla zbioru w kształcie krzyża



Wizualizacja 33: Wyniki pomiarów czasu dla zbioru w kształcie krzyża

Na podstawie **Tabeli 4** oraz wykresów z **Wizualizacji 33** można stwierdzić, że w tym przypadku **KD-Drzewo** działa zdecydowanie gorzej. Budowanie **Quad-Tree** zajmuje o 65% mniej czasu, natomiast przeszukiwanie go zajmuje ok. 68% mniej czasu. Warto również dodać, że w tym konkretnym zestawie danych liczba punktów została zmniejszona do rzędu tysięcy.
W tym konkretnym typie zbioru ukazuje się słabość **KD-Drzewa**. Z powodu tego, że wiele punktów ma równe współrzędne w danym wymiarze, wybieranie mediany za pomocą **QuickSelecta** ukwadratawia się. Mało tego, w przypadku liczby punktów większej niż 4000, każdorazowo następuje przepełnienie stosu rekurencyjnego. Z tego też powodu specjalnie na potrzebę utworzenia ładnych wykresów zwiększona została

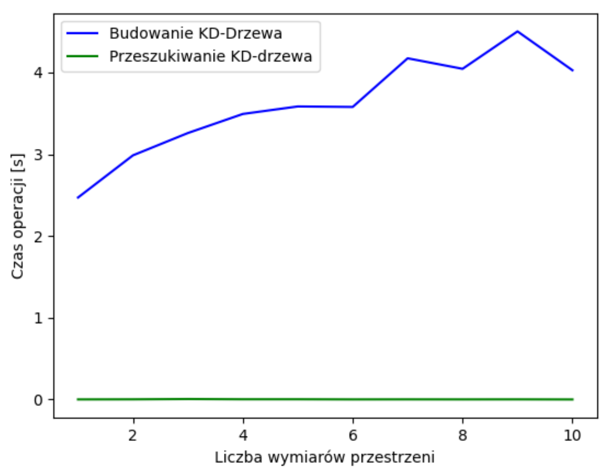
maksymalna głębość stosu. Na wykresie opisującym budowanie drzewa widać, że czas nie zwiększa się już w sposób liniowy, a przyrost się sukcesywnie zwiększa.

2.5 Testowanie KD-Tree w wielowymiarowej przestrzeni

Drzewo k-wymiarowe, jak jego nazwa wskazuje, oferuje operacje na punktach określonych nie tylko w przestrzeni dwuwymiarowej, ale również wyższych. Drzewo zostało zaimplementowane w taki sposób, aby liczby wymiarów nie miała dużego wpływu na czas działania algorytmów. W szczególności inicjalizacja drzewa ma złożoność $O(nlogn)$, inne implementacje mają złożoność silnie zależną od liczby wymiarów $O(knlogn)$. Poniżej przedstawione są dane na temat czasu działania drzewa dla 50 000 punktów.

Liczba wymiarów	Czas tworzenia KD-Tree [s]	Czas przeszukiwania KD-Tree [s]
1	2.47	0.001
2	2.986	0.002
3	3.259	0.006
4	3.492	0.003
5	3.584	0.003
6	3.578	0.001
7	4.173	0.002
8	4.044	0.001
9	4.502	0.002
10	4.026	0.0

Tabela 5: Wyniki pomiarów czasu dla wielowymiarowej przestrzeni, **KD-Tree**



Wizualizacja 34: Zależność działania **KD-Tree** od liczby wymiarów

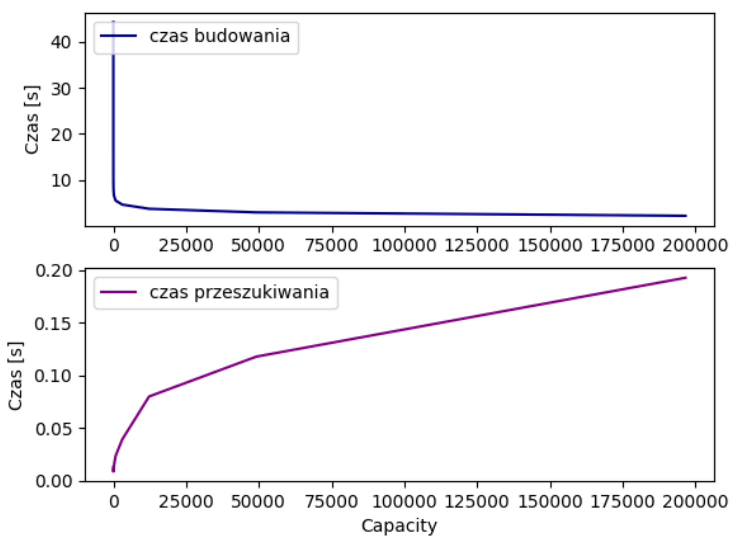
W **Tabeli 5** oraz na wykresie w **Wizualizacji 34** można dostrzec tendencję rosnąca w przypadku czasu potrzebnego na zbudowanie **KD-Tree**. Jest ona jednak niewielka i pomijalna. Przykładowo czas wykorzystany w przypadku 2 wymiarów wzrósł o niecałe 40%.

2.6 Testowanie Quad-Tree dla różnych wartości capacity

Capacity jest istotnym parametrem w procedurze budowania drzewa ćwiartkowego. Im bardziej zwiększymy jego wartość tym płytsze będzie powstałe drzewo. Dzieje się tak ze względu na zwiększanie pojemności liści, przez co mniej podziałów musi zajść, aby wybudować drzewo. Im mniej zachodzi podziałów, tym płytsze jest drzewo. Ma to jednak swoje wady. W przypadku, gdy parametr **capacity** byłby równy n, to faktycznie budowa drzewa byłaby zakończona w czasie liniowym, ponieważ wszystkie punkty zostałyby wstawione do korzenia. Jednakże, na tak skonstruowanym drzewie, algorytm przeszukiwania byłby zmuszony do rozważenia każego punktu z osobna w kontekście jego należenia do przeszukiwanego obszaru, co jest tożsame z podejściem trywialnym opisanym w następnej sekcji (**2.7**). Przeprowadzono testy czasowe dla różnych wartości **capacity**. Za zbiór testowy przyjęto 400 000 punktów generowanych według rozkładu jednostajnego. Dane są zamieszczone w poniższej tabeli.

Capacity	Czas tworzenia Quad-Tree [s]	Czas przeszukiwania Quad-Tree [s]
1	44.276	0.013
2	16.283	0.012
3	14.026	0.010
12	9.816	0.009
48	8.063	0.010
192	6.679	0.014
768	5.519	0.023
3072	4.635	0.039
12288	3.726	0.080
49152	2.952	0.118
196608	2.209	0.193

Tabela 6: Wyniki pomiarów czasu dla różnych wartości **capacity**, **Quad-Tree**

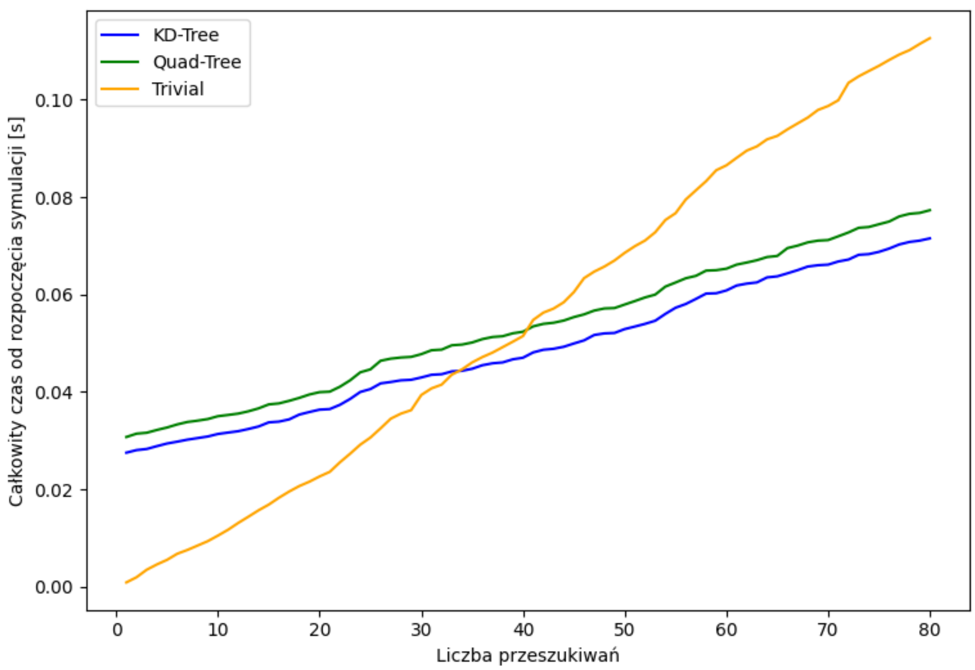


Wizualizacja 35: Zależność budowania i przeszukiwania Quad-Tree w zależności od **capacity**

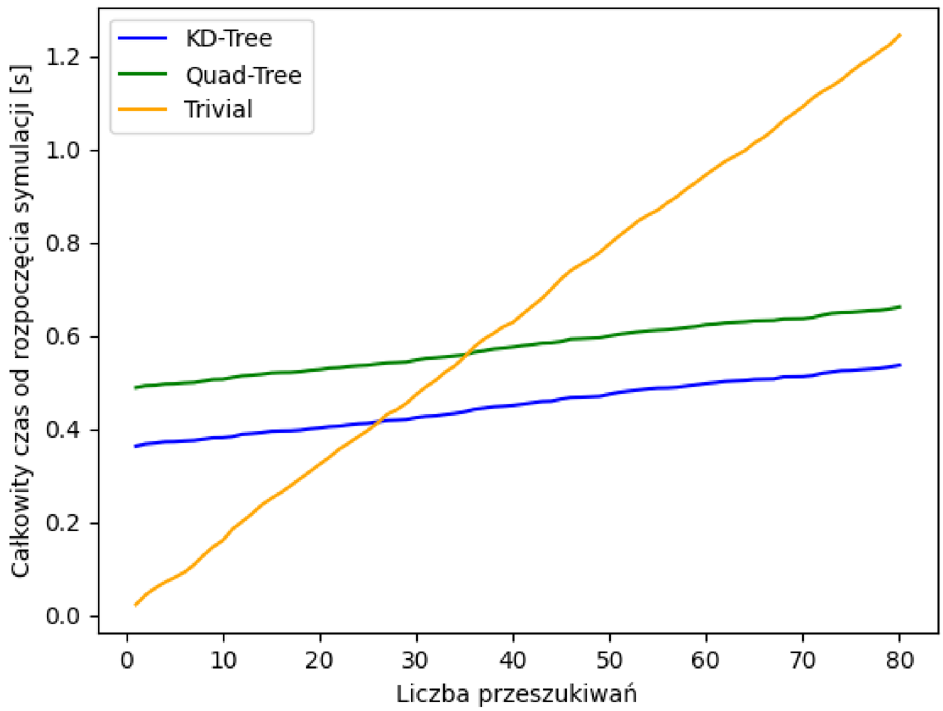
Jak widać z wyników w **Tabeli 6** oraz **Wizualizacji 35**, dla dużych wartości parametru **capacity** istotnie maleje czas budowania drzewa, ale rośnie czas przeszukiwania. Czas przeszukiwania przy **capacity** równym 196600 ($\approx \frac{n}{2}$) jest nadal mały, choć w porównaniu z **capacity** równym 1 jest to róznica niemal 15-krotna. Miałoby to jeszcze większe znaczenie w przypadku wielokrotnego przeszukiwania obszaru, co jest szerzej omówione w sekcji **2.7**.

2.7 Porównanie z podejściem trywialnym

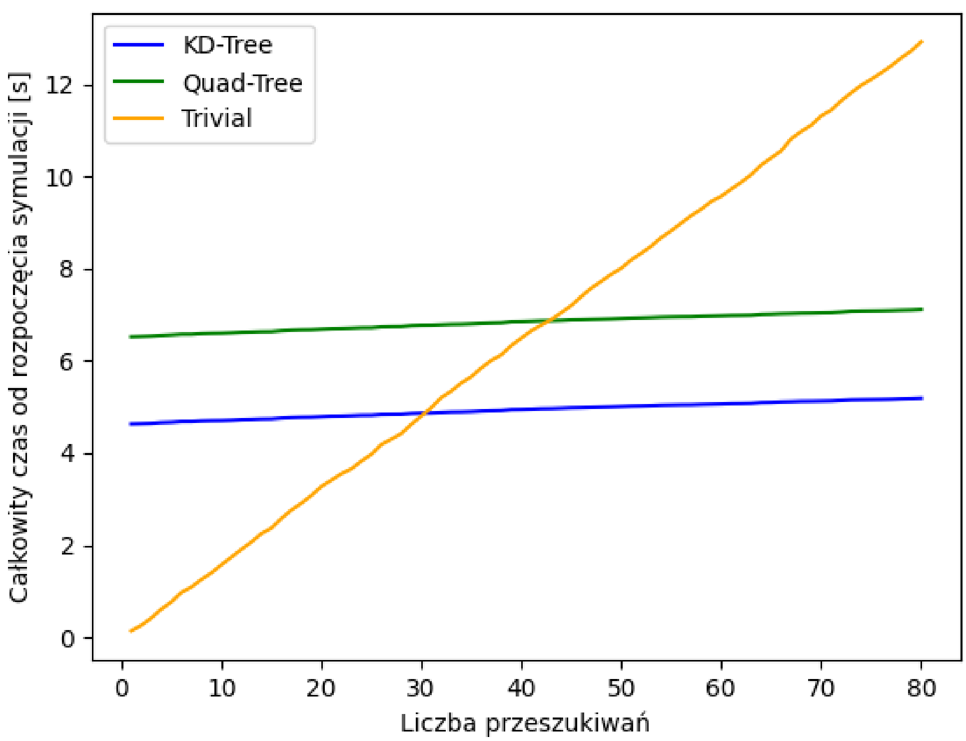
Uznaliśmy, iż ciekawym pomysłem będzie sprawdzenie, kiedy tak naprawdę zaimplementowane struktury będą opłacalne do zastosowania. Oczywistym jest, że w sytuacji kiedy jednorazowo chcielibyśmy w danym zbiorze punktów przeszukać konkretny obszar znacznie prostszym i wydajniejszym sposobem będzie zwykłe sprawdzenie każdego punkta z osobna. Takie trywialne podejście ma złożoność O(n). Natomiast w sytuacji, w której wielokrotnie przeszukiwałoby się różne obszary jednorazowe zbudowanie drzewa powinno się spłacić. Złożoność takich operacji przykładowo dla **KD-Drzewa** wynosi $O(nlogn + k\sqrt{n})$, gdzie n to liczebność zbioru, a k określa ilość przeszukiwań. Z kolei trywialne podejście wymaga dokładnie kn operacji.
Poniżej zamieszczone zostały wykresy dla różnych liczebności zbioru punktów o rozkładzie jednostajnym.



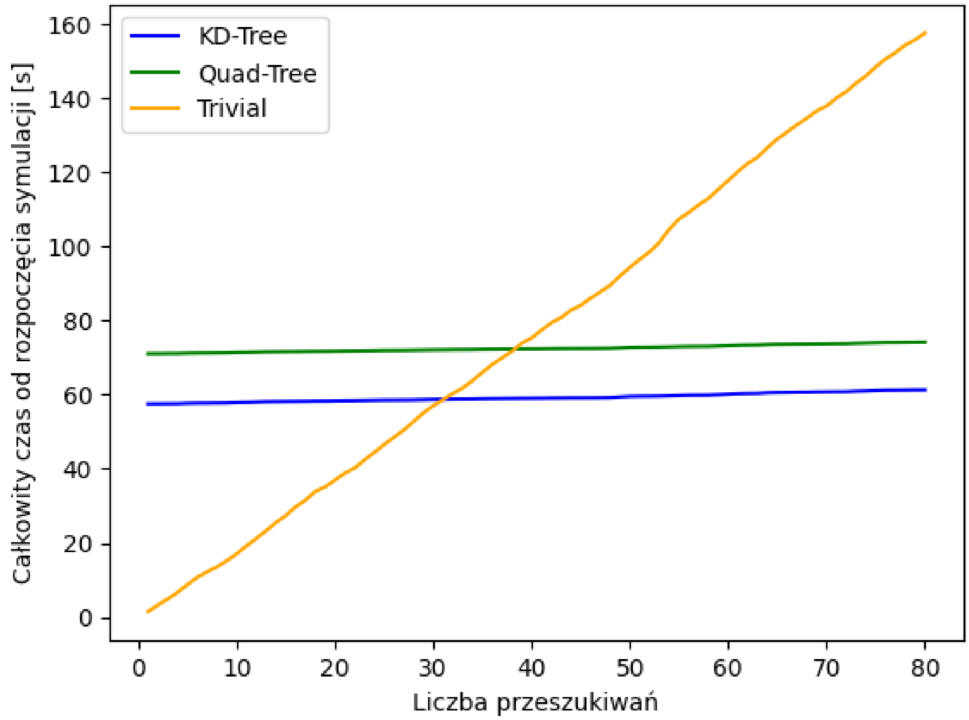
Wizualizacja 36: Wyniki dla zbioru 1000 punktów



Wizualizacja 37: Wyniki dla zbioru 10^4 punktów



Wizualizacja 38: Wyniki dla zbioru 10^5 punktów



Wizualizacja 39: Wyniki dla zbioru 10^6 punktów

Jak widać na **Wizualiacjiach 36-39**, niezależnie od liczby punktów, w okolicy 30-40 przeszukiwania drzew, używanie tych struktur staje się wydajniejsze.

Część V
Wnioski

Na podstawie testów przeprowadzonych powyżej, można stwierdzić, że struktury i algorytymy zostały poprawnie zaimplementowane oraz zwracają poprawne podzbiory punktów w zadanych obszarach prostokątnych. Dla niemalże każdego testu czas budowania **KD-Tree** był szybszy, natomiast ta struktura nie radzi sobie w sytuacjach, gdy wiele punktów posiada te same współrzędne. Oba drzewa bardzo sprawnie radzą sobie z wyszukiwaniem punktów z niewielką przewagą po stronie **Quad-Tree**. W przypadku, gdy jesteśmy pewni, że punkty mają zróżnicowane wartości warto użyć **KD-Tree**. Szczególnie w zbiorach punktach o rozkładzie podobnym do normalnego - wtedy przewaga **KD-Tree** była najwidoczniejsza. Natomiast w przeciwnych przypadkach bezpieczniejsze jest używanie **Quad-Tree**, gdyż jest uniwersalniejsze - nie ma żadnych specjalnych sytuacji, z którymi by sobie nie poradziło. Mowa oczywiście o sytuacjach, gdy zadany zbiór punktów jest wykorzystywany wielokrotnie. Dla pojedynczych operacji skuteczniejsze jest trywialne podejście o złożoności liniowej.